深度学习方法与实践第六次作业

姓名：杨玉雷
学号：18023040

1 基础作业要求

卷积可视化：
在 Lenet 中，分别使用 ReLU 及 sigmoid 激活函数，观察不同情况下，Lenet 学习 MNIST 分类时，参数的变化。
并在最终训练好 Lenet 的情况下，观察分类操作前的最后一个全连接层 fc2 的 84 位特征向量，比较不同类型样本的 fc2 特征图。

要求：提交代码，文档。文档包括可视化截图。
（1） tensorboard 可视化包括：loss， acc， w、b 参数的分布， 卷积核、全连接矩阵的参数可视化，至少比较 2 种情况：[ReLU, sigmoid] 。

有兴趣可以尝试分别使用 GradientDescentOptimizer 和 AdamOptimizer 的效果。

（2）fc2 特征图不用 tensorboard 显示，plot 绘出即可：
在模型训练好的情况下：绘制 10 个数字类型的 fc2 特征图，每个类型一张 fc2 图，共 10 张 fc2 图：
每一张 fc2 图由同类型的 100 个不同样本的 fc2 特征按行拼接组成。
例如数字 3 的 fc2 特征图，由 100 个不同的数字 3 样本的 fc2 特征按行拼接组成。
故一张 fc2 图的大小为：100（行）*84（列）。

2 实验过程

在作业四中，已实现了 Lenet 网络对 Mnist 数据集分类，在此基础上加上 tensorboard 可视化核心代码如下:

2.1 激活函数为 ReLU 和 sigmoid

在 lenet.py 文件中修改：

```
with tf.variable_scope(scope_name) as scope:
    conv_weights = tf.get_variable(name=W_name, shape=filter_shape,
initializer=tf.truncated_normal_initializer(stddev=self.sigma))
    conv_biases = tf.get_variable(name=b_name, shape=b_shape,
initializer=tf.constant_initializer(0.1))
    conv = tf.nn.conv2d(x, conv_weights, strides=conv_strides,
padding=padding_tag)
    if self.activation_function=="relu":
      act=tf.nn.relu(tf.nn.bias_add(conv, conv_biases))
    else:
        act=tf.nn.sigmoid(tf.nn.bias_add(conv, conv_biases))
    tf.summary.histogram(W_name, conv_weights)
    tf.summary.histogram(b_name, conv_biases)

with tf.variable_scope(scope_name) as scope:
    fc_weights = tf.get_variable(W_name,W_shape,
initializer=tf.truncated_normal_initializer(stddev=self.sigma))
    fc_biases = tf.get_variable(b_name,b_shape,
initializer=tf.constant_initializer(0.1))
    if self.activation_function=="relu":
          act = tf.nn.relu(tf.matmul(x, fc_weights) + fc_biases)
    else:
          act = tf.nn.sigmoid(tf.matmul(x, fc_weights) + fc_biases)
    tf.summary.histogram(W_name, fc_weights)
    tf.summary.histogram(b_name, fc_biases)

def _compute_loss_graph(self):
     cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=self.y,
logits=self.digits)
     self.loss = tf.reduce_mean(cross_entropy)
     tf.summary.scalar("loss", self.loss)

def _compute_acc_graph(self):
     self.prediction = tf.equal(tf.argmax(self.digits, 1),tf.argmax(self.y,
1))
     self.accuracy = tf.reduce_mean(tf.cast(self.prediction, tf.float32))
     tf.summary.scalar("accuracy", self.accuracy)
```

在 train.py 中修改：

```
for activation_function in ["relu","sigmoid"]:
    tf.reset_default_graph()
    lenet5 = Lenet(0.001, 0.1,activation_function)
    saver = tf.train.Saver()
```

```
merged_summary = tf.summary.merge_all()
init = tf.global_variables_initializer()
sess=tf.Session()
writer = tf.summary.FileWriter(LOGDIR + "/" + activation_function)
writer.add_graph(sess.graph)
sess.run(init)
.....................
```

loss，acc，w,b 参数的可视化如下：

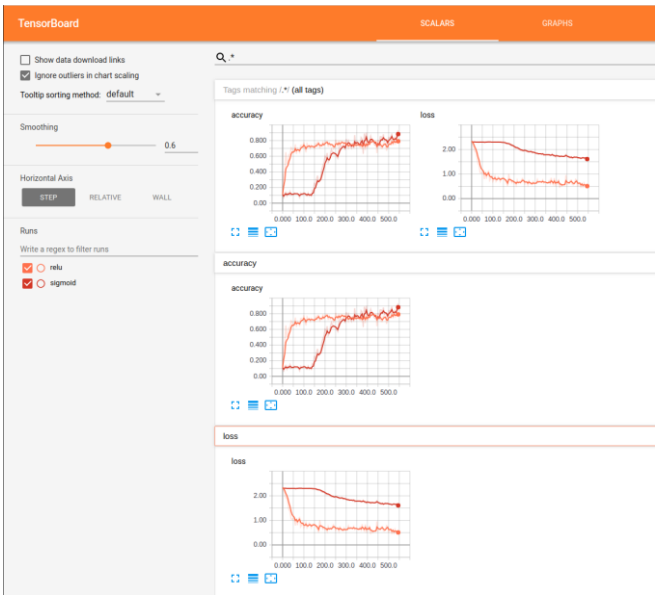


图 1 relu 和 sigmoid 下的 accuracy 和 loss 可视化

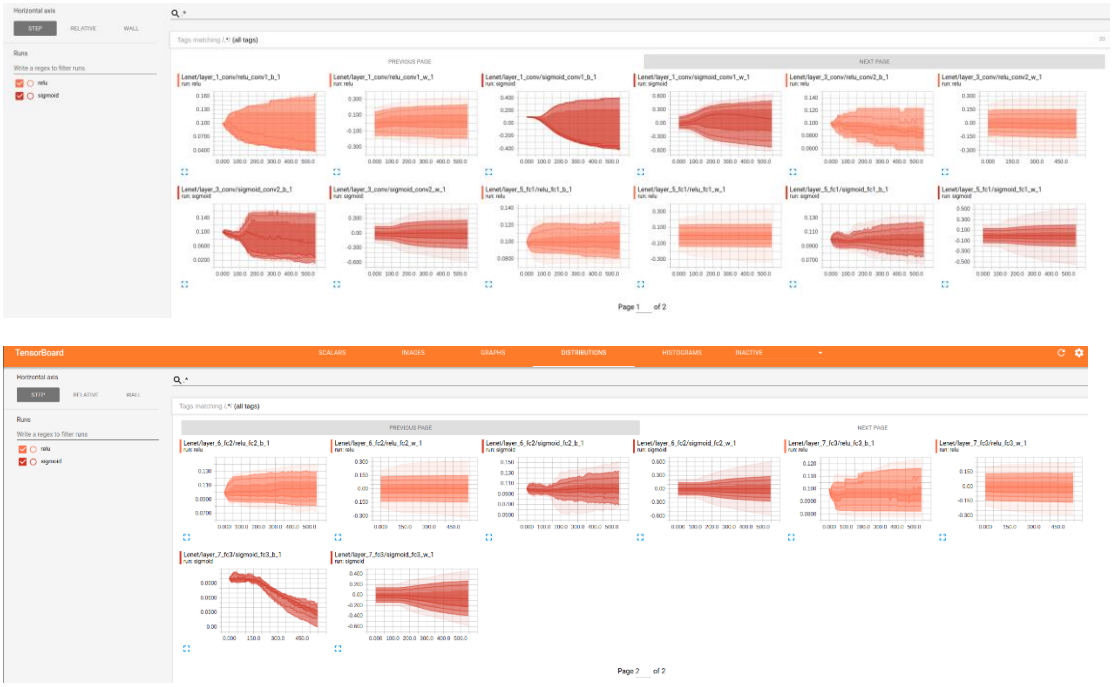
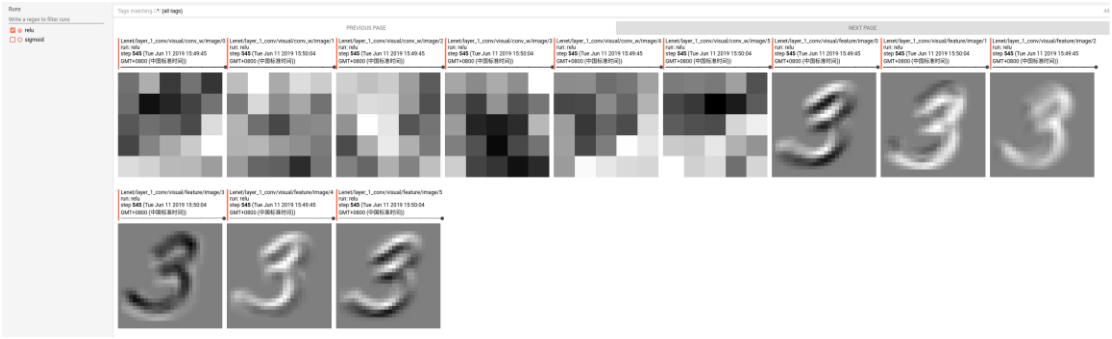


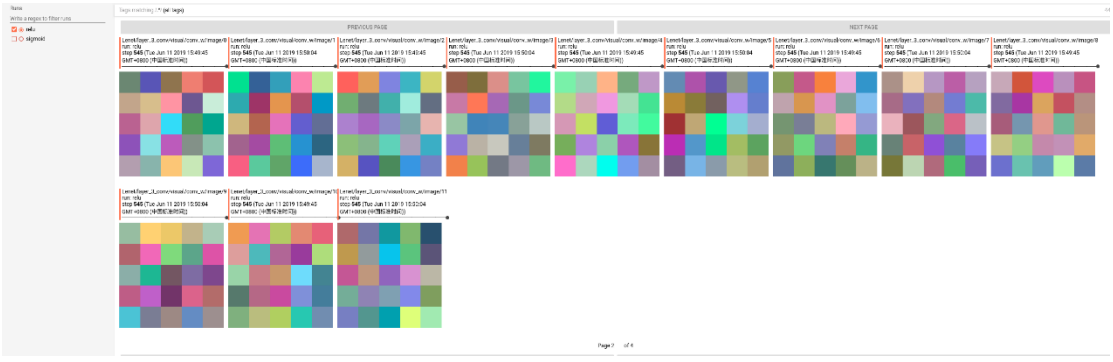
图 2 relu 和 sigmoid 下的卷积层全连接层参数 w 和 b 可视化

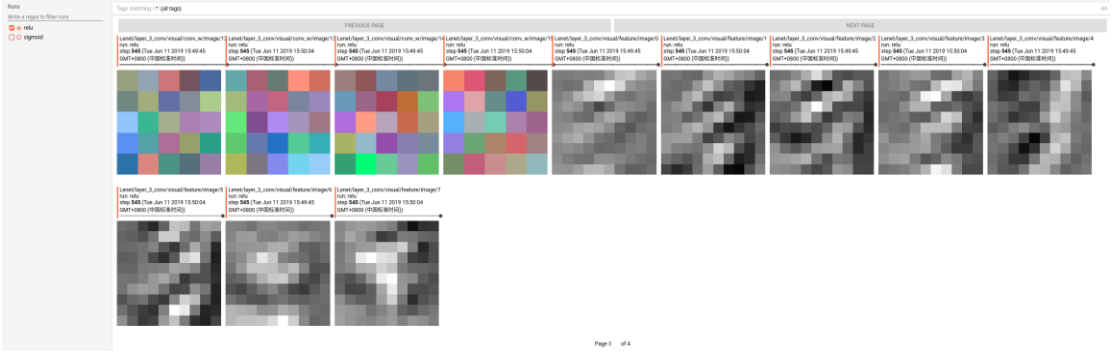
2.2 卷积核可视化

（1）合并卷积核的其中 3 个通道成为一个彩色图浏览

```
with tf.name_scope('visual') as v_s:
    x_min = tf.reduce_min(conv_weights)
    x_max = tf.reduce_max(conv_weights)
    kernel_0_to_1 = (conv_weights - x_min) / (x_max - x_min)
    kernel_transposed = tf.transpose (kernel_0_to_1, [3, 0, 1, 2])
    conv_W_img=tf.cond(tf.greater( filter_shape[2],1), lambda:
    tf.slice(kernel_transposed,[0,0,0,0],[1,filter_shape[0],filter_shape[1
    ],3]),lambda: kernel_transposed)
      tf.summary.image('conv_w', conv_W_img, max_outputs=filter_shape[3])
      feature_img = conv[0:1, :, :, 0:filter_shape[3]]
      feature_img = tf.transpose(feature_img, perm=[3, 1, 2, 0])
      tf.summary.image('feature', feature_img,max_outputs=filter_shape[3])
```







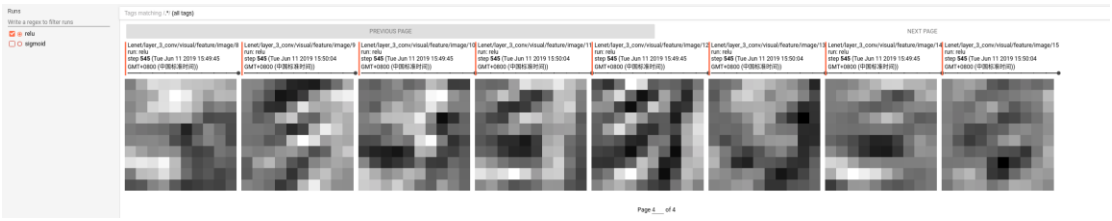
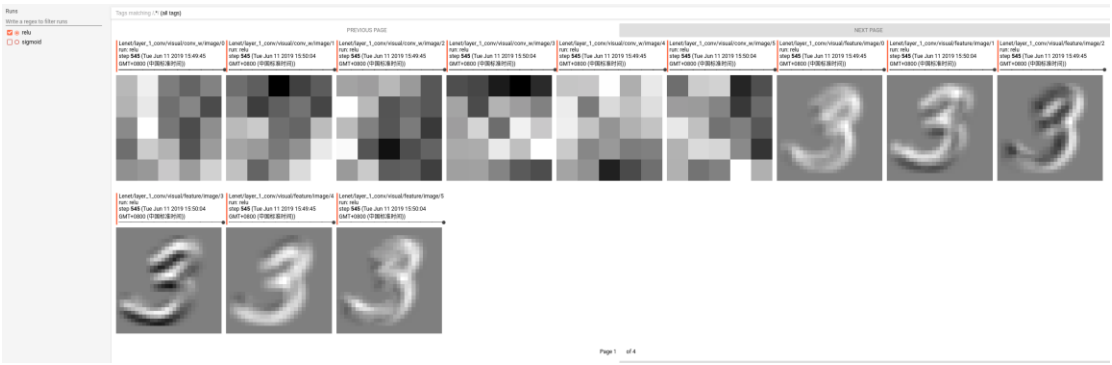


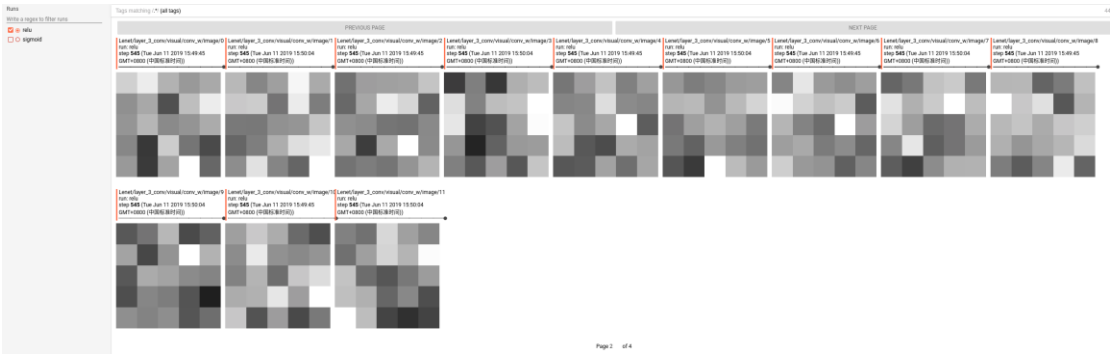
图 3 relu 下卷积核可视化

图 4 simgoid 下卷积核可视化

（2）卷积核通道单独显示

```
with tf.name_scope('visual') as v_s:
    # scale weights to [0 1], type is still float
    x_min = tf.reduce_min(conv_weights)
    x_max = tf.reduce_max(conv_weights)
    kernel_0_to_1 = (conv_weights - x_min) / (x_max - x_min)
    kernel_transposed = tf.transpose(kernel_0_to_1, [3, 2, 0, 1])
    conv_W_img = tf.reshape(kernel_transposed, [-1, filter_shape[0],
filter_shape[1], 1])
    tf.summary.image('conv_w', conv_W_img, max_outputs=filter_shape[3])
    feature_img = conv[0:1, :, :, 0:filter_shape[3]]
    feature_img = tf.transpose(feature_img, perm=[3, 1, 2, 0])
    tf.summary.image('feature', feature_img, max_outputs=filter_shape[3])
```





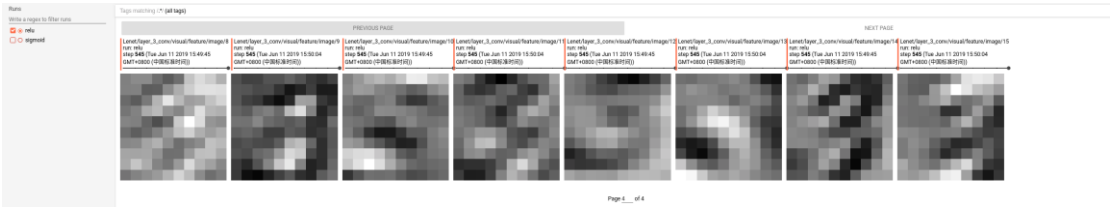


图 5 relu 下卷积核可视化

图 6 simgoid 下卷积核可视化

2.3 全连接层矩阵可视化

在 Lenet.py 的全连接层定义中加入如下代码：

```
with tf.name_scope('fc') as v_s:
    x_min = tf.reduce_min(fc_weights)
    x_max = tf.reduce_max(fc_weights)
    fc_W_img = (fc_weights - x_min) / (x_max - x_min)
    fc_W_img_reshape = tf.reshape(fc_W_img, [-1, W_shape[0], W_shape[1], 1])
    tf.summary.image(W_name, fc_W_img_reshape)
```

Lenet 的三个全连接层矩阵，size 分别为[400,120],[120,84],[84,10]转化为单通道的 Tensor 后，可视化如图 7 所示。

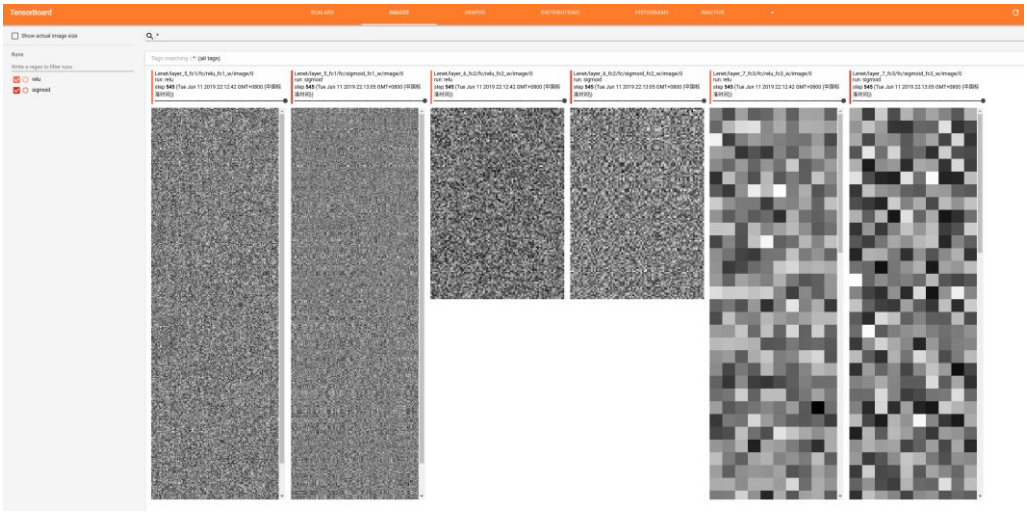


图 7 relu 和 simgoid 激活函数下的三个全连接矩阵的可视化图

2.4 fc2 特征图可视化

使用下面的方法从训练集中读取标签是 0-9 的数据集 100 行：

```
def load_test(n):
    # 不用 onehot 读取，这样 label 就是 0-9 的数
    mnist1 = read_data_sets("MNIST_data/")
    # labels:[7 3 4 ..., 5 6 8]
    labels= mnist1.test.labels

    # 一个全是 n 的一维数组:[ n.  n.  n. ...,  n.  n.  n.]
    nArray = n * np.ones(labels.shape)
    # 对比得到一个模板，用来筛选数字为 n 的图片:mask:[False  True False ...,
False False False]
    mask = np.equal(labels, nArray)

    mnist2 = read_data_sets("MNIST_data/",reshape=False,one_hot=True)
    #取出相同数字的不同测试样本图片
    X_test=mnist2.test.images[mask, :]
```

```
X_test = np.pad(X_test, ((0, 0), (2, 2), (2, 2), (0, 0)), 'constant')

y_test=mnist2.test.labels[mask, :]
#取一百行
X=X_test[:100]
y=y_test[:100]
#(100, 784)
print("x.shape",X.shape)
return X,y
```

在测试时加入如下绘图代码：

```
    saver.restore(sess,os.path.join(MODEL_SAVE_PATH, MODEL_NAME))  # 加载模型训练好的的网络和参数来测试，或进一步训练

    images={i:[] for i in range(10)}
    for i in range(10):
        X_test, y_test = load_test(i)
        # X_test, y_test = load_test(i)
        fc2=  sess.run([lenet5.fc2_relu],  feed_dict={lenet5.x:  X_test, lenet5.y: y_test})
        # print("Test Accuracy = {:.3f}".format(test_accuracy))
        fc2=(fc2-np.min(fc2))/(np.max(fc2)-np.min(fc2))
        print("fc2.shape",fc2.shape) #[100,84]
        images[i]=fc2
    for i in range(10):
        #一行 10 列
        plt.subplot(1,10,i+1)
        plt.imshow(images[i],cmap='gray')
        plt.title(i)
        plt.axis('off')
    plt.savefig(activation_function+'_test.png')
    plt.show()
```

每个数字的 FC2 特征图如图 8 和图 9 所示意。

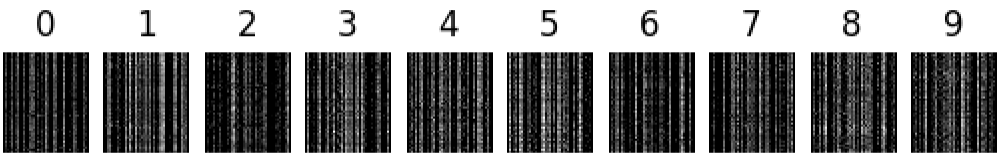


图 8 激活函数为 relu 时的 fc2 特征图

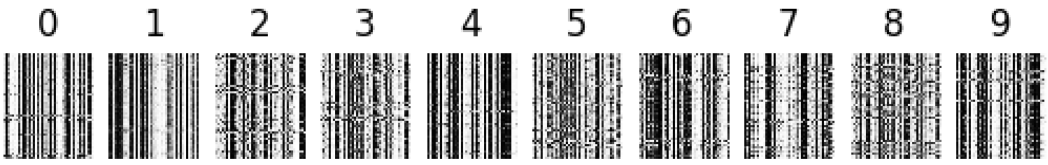


图 9 激活函数为 sigmoid 时的 fc2 特征图