

实验任务

在本实验中，我们将指导大家利用 TensorFlow 完成通用逼近定理的简单展示，即用神经网络拟合一个线性函数；同时作为练习，要在每个步骤时，完成检验学习效果的拟合非线性函数。

对于线性函数，我们的 ground-truth 模型是 $y = 1.6x + 0.4$ ，对于非线性函数，我们的 ground-truth 模型是 $y = 0.8x^2 - 1.6x + 1.2$ 。

实验预备：

1. 为方便观察，请安装结合 TensorFlow 与 matplotlib 的库 tfmpl:
C:\Users\MSUser>pip install tfmpl
2. 请在 D 盘下或其它目录下新建文件夹 lab-2

生成模拟数据

子任务：

我们考察区间 $[0, 4]$ ，我们生成区间内 100 个横坐标点，然后根据公式 $y = 1.6x + 0.4$ 生成真实标签： y^d ；然后加上适当的噪声 ϵ 后，即 $y = y^d + \epsilon$ ，来模拟未经训练的神经网络的输出。这样，后续通过观察训练前后的输出，我们可以评估神经网络的建模能力。

步骤：

1. 在 lab-2 文件夹下新建文件 lr_data.py
2. 输入以下内容：

```
import numpy as np
import tensorflow as tf
import tfmpl

if not os.path.exists("output-data"):
    os.makedirs("output-data")

x = tf.random.uniform((100, 1)) * 4
y_ = 1.6 * x + 0.4
y = y_ + tf.random.normal(tf.shape(x), stddev = 0.25)

pts = tf.concat([x, y_, y], axis = -1)

@tfmpl.figure_tensor
def draw_scatter(scaled, colors):
    """Draw scatter plots. One for each color."""
```

```
fig = tfmpl.create_figures(1, figsize = (4, 4))[0]
ax = fig.add_subplot(1, 1, 1)
ax.scatter(scaled[:, 0], scaled[:, 1], c=colors[0])
ax.scatter(scaled[:, 0], scaled[:, 2], c=colors[1])
fig.tight_layout()
return fig
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    image_tensor = draw_scatter(pts, ['r', 'b'])
    image_summary = tf.summary.image('scatter', image_tensor)
    all_summaries = tf.summary.merge_all()

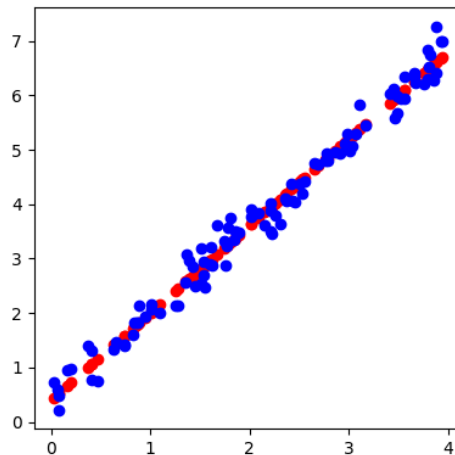
    writer = tf.summary.FileWriter('output-data', sess.graph)
    summary = sess.run(all_summaries)
    writer.add_summary(summary, global_step = 0)
```



为方便起见，文件 lr_data.py 附上： lr_data.py

- 查看输出结果，如果程序无误，则输出如下图所示：

```
D:\lab-2>python lr_data.py
tensorboard --logdir=D:\\lab-2\\output-data --port=8008
http://localhost:8008/#images
```



实作：

考察范围 $[-4, 4]$ ，根据公式 $y = 0.8x^2 - 1.6x + 1.2$ 生成真实数据 (x, y^d) ，及模拟数据 (x, y) ，并画出相应图形。

小结:

可以通过考察真实数据的分布特征, 决定所选取模型的复杂度。

建立模型

子任务:

我们用含有两个隐含层的神经网络来建模线性函数, 第一个隐含层有 16 个节点, 第二个隐含层有 8 个节点。**注意, 对于一元线性函数, 我们用没有隐含层的平凡网络就能达到这个目的, 但这里纯粹是为了学而这样设计的。**

步骤:

1. 在 lab-2 文件夹下新建文件 lr_model.py
2. 输入以下内容:

```
import numpy as np
import tensorflow as tf
import sonnet as snt

class LRModel(snt.AbstractModule):
    def __init__(self, name = "lr_model"):
        super(LRModel, self).__init__(name = name)

        with self._enter_variable_scope():
            self._h1 = snt.Linear(16)
            self._h2 = snt.Linear(8)
            self._out = snt.Linear(1)

    def _build(self, x):
        y = tf.nn.relu(self._h1(x))
        y = tf.nn.relu(self._h2(y))
        y = tf.nn.relu(self._out(y))

        return y

if __name__ == "__main__":
    import os

    if not os.path.exists("output-model"):
        os.makedirs("output-model")

    x = tf.random.uniform((32, 1)) * 4
    lr_model = LRModel()
    y = lr_model(x)
```

```

writer = tf.summary.FileWriter('output-model, tf.get_default_graph())
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    x_val, y_val = sess.run([x, y])
    print("x -> {}".format(x_val))
    print("y -> {}".format(y_val))

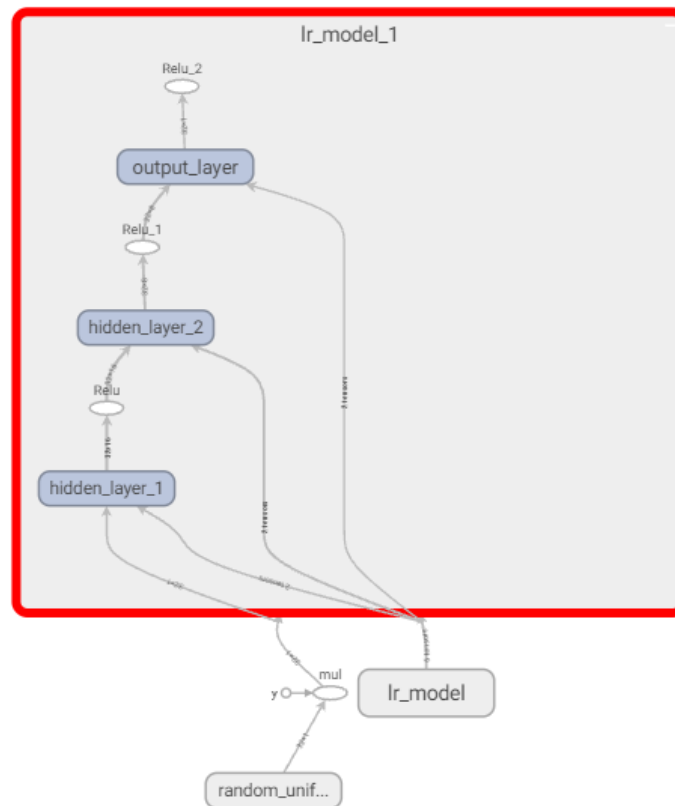
writer.close()

```



为方便起见，文件 `lr_model.py` 附后：

- 查看输出结果，如果程序无误，则输出如下图所示：
`D:\lab-2>python lr_model.py`
`tensorboard --logdir=D:\\lab-2\\output-model --port=8008`
[http://localhost:8008/#graphs&run=.](http://localhost:8008/#graphs&run=)



实作：

建立非线性函数的模型，请用三个隐含层，节点数目分别是 32， 16， 8， 通过 TensorBoard 观察所建立模型是否正确。

小结：

可以通过分析 TensorBoard 所呈现的 graph， 判断所建立的模型的正确性。

训练模型

子任务：

我们用 mini-batch 方法， 训练新建模型。

步骤：

1. 在 lab-2 文件夹下新建文件 lr_train.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf

from lr_model import LRModel

# 执行自动微分算法（或反向传播算法）的优化器参数
learning_rate = 0.1
lr_decay_steps = 100
lr_decay_factor = 0.9

# 训练时迭代次数
iterations = 1000

if not os.path.exists("output-train"):
    os.makedirs("output-train")

# 训练好的模型的保存路径
checkpoint_path = os.path.join("output-train", "model.ckpt")

# 生成样本，包含输入数据与真实标签
x = np.random.uniform(size = (100, 1)) * 4
y = 1.6 * x + 0.4
```

```

# 将输入包装成数据集，方便进行 mini-batch 训练
ds = tf.data.Dataset.from_tensor_slices(tf.concat([x, y], axis = -1))
ds = ds.shuffle(buffer_size = 64).batch(32).repeat(-1)
ds = ds.map(lambda s: tf.split(s, num_or_size_splits = 2, axis = -1))
it = ds.make_one_shot_iterator()

x, y_ = it.get_next()

# 由输入得到模型的输出
net = LRModel()
y = net(x)

# 计算损失函数
with tf.control_dependencies([tf.assert_equal(tf.rank(y), tf.rank(y_))]):
    loss = tf.reduce_mean(tf.squared_difference(y, y_), name = 'loss')

loss_summary = tf.summary.scalar('loss', loss)

# 设置学习率
global_step = tf.train.get_or_create_global_step()
lr = tf.train.exponential_decay(learning_rate, global_step,
    lr_decay_steps, lr_decay_factor, staircase = True)

lr_summary = tf.summary.scalar('lr', lr)

# 创建优化器
opt = tf.train.AdamOptimizer(lr)

# 进行优化
train_op = opt.minimize(loss, global_step = global_step, var_list =
    tf.trainable_variables())

# 合并所有 summary 信息
all_summaries = tf.summary.merge_all()

writer = tf.summary.FileWriter('output-train', tf.get_default_graph())

# 保存训练好的模型
saver = tf.train.Saver(tf.trainable_variables())

# 创建 Session，进行训练
with tf.Session() as sess:
    sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])

```

```

for i in range(iterations):
    loss_val, _, summ_str = sess.run([loss, train_op, all_summaries])
    print("{}-th iteration with loss {}".format(i, loss_val))
    writer.add_summary(summ_str, global_step = i)

# 训练完成，保存模型
print('Saving model.')
saver.save(sess, checkpoint_path)
print('Training complete')

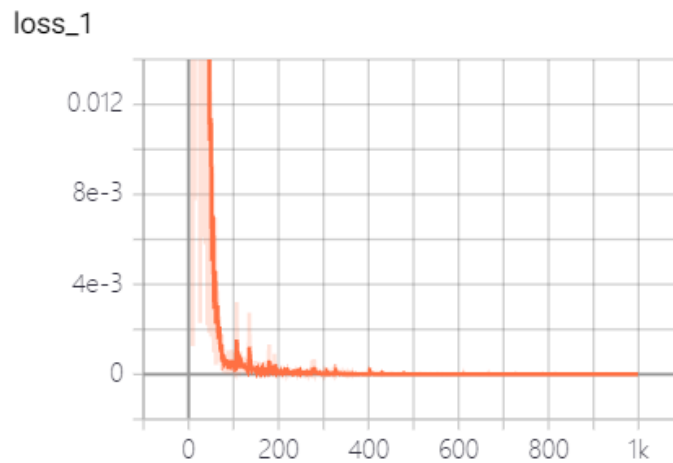
writer.close()

```



为方便起见，文件 lr_train.py 附上： lr_train.py

- 查看输出结果，如果程序无误，则输出如下图所示：
D:\lab-2>python lr_train.py
tensorboard --logdir=D:\\lab-2\\output-train --port=8008
<http://localhost:8008/#scalars>



实作：

依据上节中所建网络，训练非线性函数对应的网络模型

测试模型

子任务：

测试我们用 mini-batch 方法所训练模型的表现。

步骤：

1. 在 lab-2 文件夹下新建文件 lr_test.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf
import tfmpl

from lr_model import LRModel

if not os.path.exists("output-test"):
    os.makedirs("output-test")

if not os.path.exists("output-train"):
    raise ValueError("Non-existing output-train folder")

# 训练好的模型的保存路径
checkpoint_dir = "output-train"

# 生成测试数据
x = tf.random.uniform((100, 1), dtype=tf.float64) * 4 + 2.0
y_ = 1.6 * x + 0.4

# 由输入得到模型的输出
net = LRModel()
y = net(x)

pts = tf.concat([x, y_, y], axis = -1)

@tfmpl.figure_tensor
def draw_scatter(scaled, colors):
    """Draw scatter plots. One for each color."""
    fig = tfmpl.create_figures(1, figsize = (8, 4))[0]
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.scatter(scaled[:, 0], scaled[:, 1], c=colors[0])
    ax2 = fig.add_subplot(1, 2, 2)
```



```
ax2.scatter(scaled[:, 0], scaled[:, 2], c=colors[1])
fig.tight_layout()
return fig
```

```
saver = tf.train.Saver(tf.trainable_variables())
```

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

```
    sess.run([tf.global_variables_initializer(), tf.local_variables_initializer()])
```

```
    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
```

```
    if ckpt and ckpt.model_checkpoint_path:
```

```
        # Restores from checkpoint.
```

```
        saver.restore(sess, ckpt.model_checkpoint_path)
```

```
        print('Successfully loaded model from %s.' % ckpt.model_checkpoint_path)
```

```
    else:
```

```
        print('No checkpoint file found')
```

```
        exit
```

```
    image_tensor = draw_scatter(pts, ['r', 'b'])
```

```
    image_summary = tf.summary.image('scatter', image_tensor)
```

```
    all_summaries = tf.summary.merge_all()
```

```
    writer = tf.summary.FileWriter('output-test', sess.graph)
```

```
    summary = sess.run(all_summaries)
```

```
    writer.add_summary(summary, global_step = 0)
```



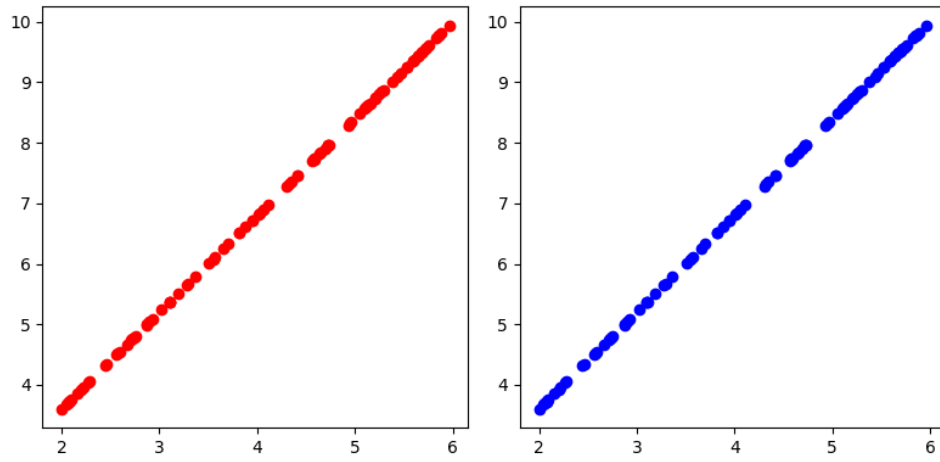
为方便起见，文件 lr_test.py 附后： lr_test.py

3. 查看输出结果，如果程序无误，则输出如下图所示：

```
D:\lab-2>python lr_test.py
```

```
tensorboard --logdir=D:\\lab-2\\output-test --port=8008
```

[http://localhost:8008/#images&run=.](http://localhost:8008/#images&run=)



实作:

依据上节训练结果，测试所训练的非线性函数的模型的表现，区间选择为 $[-6, 6]$ 。