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（Universal Approximation Theorem）的简单展示，即用神经网络拟合一个线性函数或线性模型；同时作为作业，完成对更为复杂的非线性函数或非线性模型的拟合。

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$y = 0.8x^2 - 1.6x + 1.2$。

零、实验预备：

1. 保证实作零中所有步骤已经完成，特别如下三个库已经安装： TensorFlow，Sonnet， tfmpl；否则，请回到实作 1，保证三个库已经全部成功安装；
2. 请在 D 盘下或其它目录下新建文件夹，命名为 assignment-1 或其他名称；
3. 打开包含 Python 解释器的命令行，并切换到新建的文件夹下。

一、生成模拟实际问题数据

我们考察区间$[0, 4]$，假设我们采集的样本集共有 100 个坐标点。我们首先生成区间内 100 个横坐标点，然后根据公式$y = 1.6x + 0.4$，得到 ground-truth 模型下的值。但通常在实际问题中，不仅我们对 ground-truth 模型是不知道的，并且相关的数据都是测量得来的，可能会有噪声，与真实数据有一定程度的差异。因此，我们最后加上适当的噪声ϵ后，即$y = y^d + \epsilon$，来模拟实际问题中的数据。然后通过观察这些数据的特征，来猜测模型。

步骤：

1. 在当前目录下新建文件 lr_data.py
2. 输入以下内容：

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

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

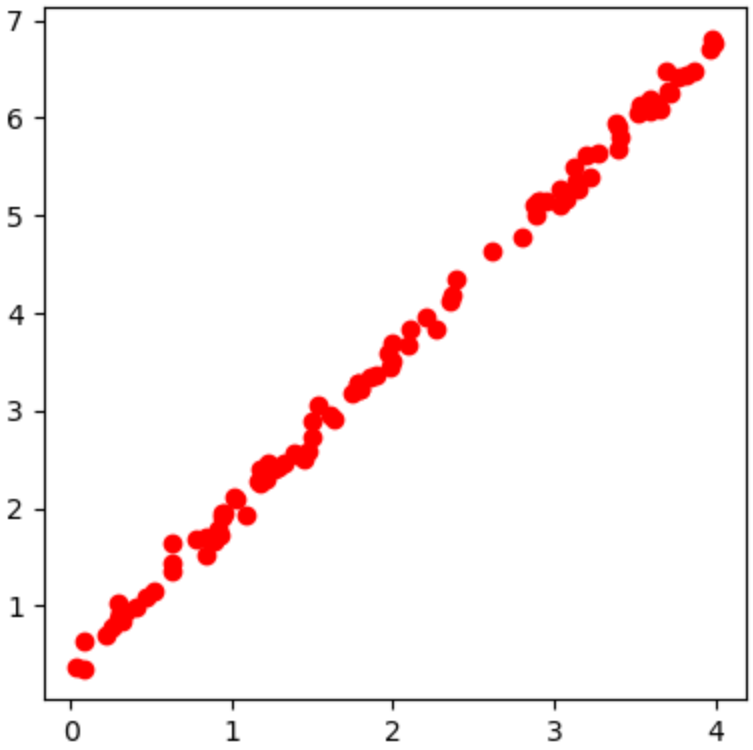
```
# 生成区间[0, 4]之内的数据
# y_d 表示真实模型的数据，通常我们并不知道；
# y 表示带噪声或实际问题中的数据，我们经过采集或采样得到；
x = tf.random.uniform((100, 1)) * 4
y_d = 1.6 * x + 0.4
y = y_d + tf.random.normal(tf.shape(x), stddev=0.1)

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

@tfmpl.figure_tensor
def draw_scatter(scaled, colors):
    '''Draw scatter plots. One for each color.'''
    fig = tfmpl.create_figure(figsize=(4, 4))
    ax = fig.add_subplot(1, 1, 1)
    for i,c in enumerate(colors):
        ax.scatter(scaled[:, 0], scaled[:, i+1], c=c)
    fig.tight_layout()
    return fig

summary_writer = tf.summary.create_file_writer(log_dir)
with summary_writer.as_default():
    image_tensor = draw_scatter(pts, ['r'])
    image_summary = tf.summary.image("images", image_tensor, step=0)
summary_writer.close()
```

当逼近其它函数时，需要将此更改为对应的函数

为方便起见，文件 lr_data.py 附后：

lr_data.py

3. 在命令行执行命令：python lr_data.py
   要确保除了 warning 之外，没有 error，否则代表没有执行成功，需要查找具体原因；
4. 查看执行输出，如果程序无误，则在命令行执行如下命令启动 TensorBoard：
   tensorboard --logdir=output-data
5. 查看数据情况，打开浏览器，键入地址：127.0.0.1:6006，则可以看到页面包含如下图形：



小结:

可以通过考察真实数据的分布特征，确定所选取模型与其复杂度。

实作:

考察范围$[-4,4]$，根据公式$y=0.8x^2-1.6x+1.2$，修改上面代码，生成真实数据(x,y^d)，及模拟数据(x,y)，修改的代码附在下面，并将浏览器中呈现数据分布的图贴在下面。

二、建立模型

子任务:

在任务一中，通过观察数据的特征，我们发现其在一条直线上分布，是线性模型。注意，对于一元线性函数，我们用单个神经元，即没有隐含层的平凡网络就能达到这个目的，就能很好地拟合网络，但这里纯粹是为了学而设计成全连接网络的。

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。

步骤：

1. 在当前文件夹下新建文件 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.Module):
    def __init__(self, name = "lr_model"):
        super(LRModel, self).__init__(name = name)

    @snt.once
    def _initialize(self):
        self._h1 = snt.Linear(16, name = "hidden_layer_1")
        self._h2 = snt.Linear(8, name = "hidden_layer_2")

        self._out = snt.Linear(1, name = "output_layer")

    def __call__(self, x):
        self._initialize()
        y = tf.nn.relu(self._h1(x))
        y = tf.nn.relu(self._h2(y))

        y = self._out(y)

        return y

if __name__ == "__main__":
    import os

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

    model = tf.function(LRModel())
    x = tf.random.uniform((32, 1)) * 4
    summary_writer = tf.summary.create_file_writer(log_dir)

    tf.summary.trace_on(graph=True, profiler=False)
```

当逼近非函数时，如果需要增加层数，请在这里先定义，如：

self._h3 = snt.Linear(8, name = "hidden_layer_3")

如果定义了新的层，请在这里建构在一起，如 y = tf.nn.relu(self._h3(y))，表示上层的输出经过本层之后，再施以激活函数 ReLU

```
y = model(x)
with summary_writer.as_default():
    tf.summary.trace_export(name="model_trace", step=0, profiler_outdir=log_dir)
tf.summary.trace_off()
```

为方便起见，文件 lr_model.py 附后：

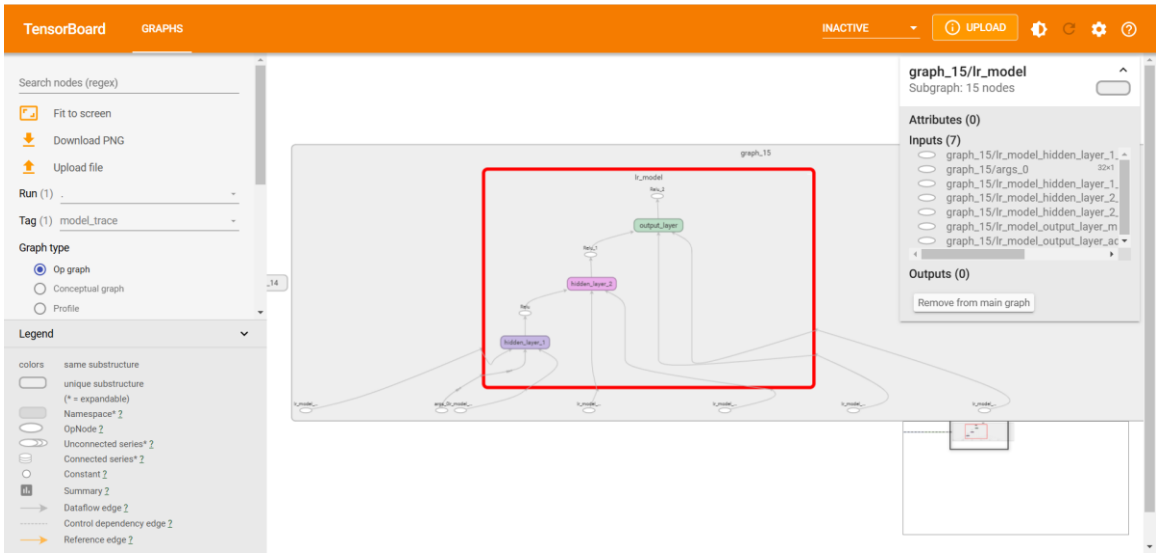
lr_model.py

3. 在命令行执行脚本文件：python lr_model.py

4. 如果程序无误，查看输出结果，则可以通过在命令行中执行如下命令启动 TensorBoard：

   tensorboard --logdir= output-model --port=8008

   然后在浏览器中键入地址：http://localhost:8008，将最后一个节点打开，看到如下所示的模型结构图：



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。可以通过运行 tensorboard 命令时的 port 参数改变 tensorboard 服务的端口。通常，完整的模型结构信息通常包含在如下图所示的最后一个节点中：

实作：

建立模拟非线性函数的模型，请用三个隐含层，节点数目分别是32， 16， 8，请将修改的代码贴在下面，并通过 TensorBoard 观察所建立模型是否正确，且将浏览器中呈现的模型结构图片贴在下面。

训练模型

子任务：

我们用 mini-batch 方法，训练新建模型。

步骤：

1. 在当前文件夹下新建文件 lr_train.py
2. 输入以下内容

```
import os
import numpy as np
import tensorflow as tf
import sonnet as snt
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 = 0.01

# 记录训练信息
```

如果观察到训练不收敛，可能学习率设置过大（或过小），请调整学习率

```
log_dir = "output-train"
if not os.path.exists(log_dir):
    os.makedirs(log_dir)
summary_writer = tf.summary.create_file_writer(log_dir)

# 因为问题比较简单，直接生成训练与验证样本
# 包含输入数据与标签，标签加噪声模拟实际情况
x_train = np.random.uniform(size=(100, 1)) * 4
y_train = 1.6 * x_train + 0.4 + tf.random.normal(tf.shape(x_train), stddev=0.1)

# 将输入包装成数据集与输入管线，方便进行 mini-batch 训练
ds_train = tf.data.Dataset.from_tensor_slices(tf.concat([x_train, y_train], axis=-1))
ds_train = ds_train.shuffle(buffer_size = 100).batch(32).repeat(1)
ds_train = ds_train.map(lambda s: tf.split(s, num_or_size_splits=2, axis=-1))

x_val = np.random.uniform(size=(40, 1)) * 4
y_val = 1.6 * x_val + 0.4 + tf.random.normal(tf.shape(x_val), stddev=0.1)

# 将输入包装成数据集与输入管线，方便进行验证
ds_val = tf.data.Dataset.from_tensor_slices(tf.concat([x_val, y_val], axis=-1))
ds_val = ds_val.batch(16).repeat(-1)
ds_val = ds_val.map(lambda s: tf.split(s, num_or_size_splits=2, axis=-1))

# 建立模型与优化器
model = LRModel()
opt = snt.optimizers.Adam(learning_rate)

# 定义单步训练
def train(x, labels, step):
    # 记录前向传播信息
    with tf.GradientTape() as tape:
        y = model(x)
        loss = tf.math.reduce_mean(
            tf.math.squared_difference(y, labels))

    # 计算梯度，并更新权重
    variables = model.trainable_variables
    gradients = tape.gradient(loss, variables)
    opt.apply(gradients, variables)
```

当逼近其它函数时，需要将此更改为对应的函数

当逼近其它函数时，需要将此更改为对应的函数

```
    with summary_writer.as_default():
        tf.summary.scalar('loss', loss, step=step)

    return loss

# 验证模型的超参
def validate(dataset_iter, step):
    x, labels = next(dataset_iter)
    y = model(x)
    err = tf.math.reduce_mean(
        tf.math.squared_difference(y, labels))

    with summary_writer.as_default():
        tf.summary.scalar('error', err, step=step)

    return err

# 保存模型
def save_model(module):
    feature_size = 1
    @tf.function(input_signature=[tf.TensorSpec([None, feature_size])])
    def inference(x):
        return module(x)

    save_path = r"saved_model"
    if not os.path.exists(save_path):
        os.makedirs(save_path)

    to_save = snt.Module()
    to_save.inference = inference
    to_save.all_variables = list(module.variables)

    tf.saved_model.save(to_save, save_path)

# 训练模型
@tf.function
def main():
    num_epochs = 20
    step = np.int64(0)
    it_val = iter(ds_val)
```

```
    for _ in range(num_epochs):
        for x, labels in ds_train:
            step = step + 1
            loss = train(x, labels, step)
            err = validate(it_val, step)
            tf.print("iteration:", step, " loss - ", loss, " error - ", err)

if __name__ == "__main__":
    main()

    # 保存模型
    save_model(model)
```

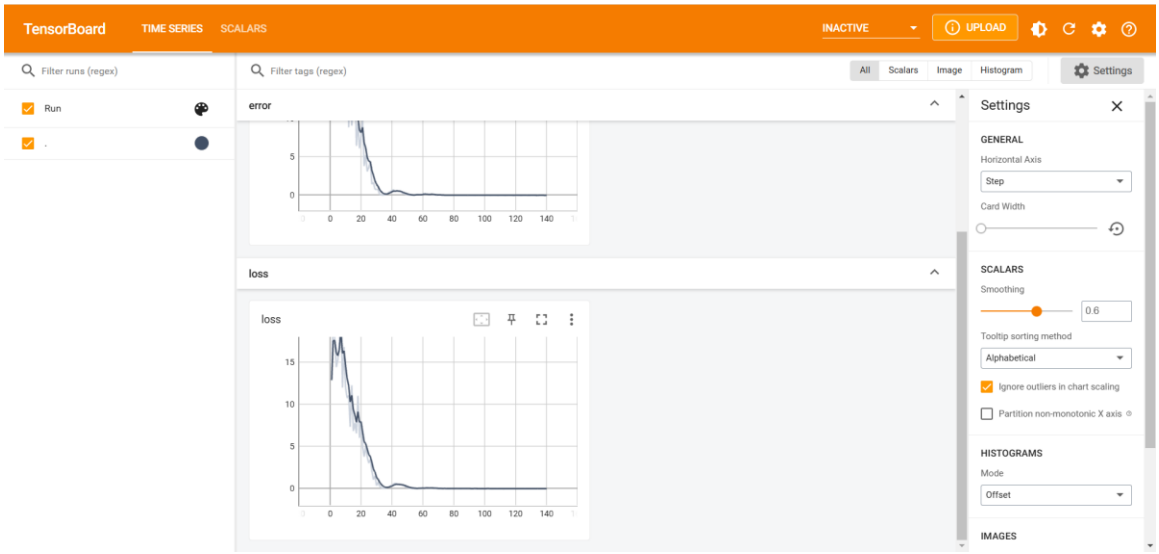
为方便起见，文件 lr_train.py 附后：

lr_train.py

3. 在命令行执行如下命令，训练网络：
   python lr_train.py

4. 如果命令行执行无误，在训练完成后，启动 TensorBoard 查看训练情况：
   tensorboard --logdir=output-train

5. 打开浏览器，键入网址 127.0.0.1:6006，可以看到如下类似的训练过程训练集上的损失数值与验证集上错误数值减小的情况：



小结:

从训练过程的指标记录上可以看出，无论时训练集上的 loss 值，与验证集上的 error 值，都收敛到较小的数值，且收敛趋势一致。说明搭建的神经网络模型能较好地适配数据，解决实际问题。

实作:

依据本任务示例代码与上节中所建网络，训练建模非线性函数所对应的网络模型。将修改的代码贴在下面，同时将 TensorBoard 呈现在浏览器中关于训练过程的图像贴在下面。

测试模型

子任务:

测试我们训练完毕的模型的表现。

步骤:

1. 在当前文件夹下新建文件 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

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

# 训练好的模型的保存路径
save_path = "saved_model"
if not os.path.exists(save_path):
    raise ValueError("Cannot find saved model")

# 生成测试数据，为使对比更加明显，
# 我们假设数据不含噪声。
x = tf.random.uniform((100, 1)) * 4 + 2.0
```

如果要变更测试时 x 的区间，请在这里更改。注意 tf.random.uniform 生成[0, 1]之间的随机数，因此生成[a, b]区间的随机数时，需要做适当的变换。

```
y_d = 1.6 * x + 0.4
```

当逼近其它函数时，需要将此更改为对应的函数

```
# 模型加载函数
def load_model(save_path):
    assert os.path.exists(save_path), "模型路径不存在"

    loaded = tf.saved_model.load(save_path)
    assert len(loaded.all_variables) > 0, "加载模型失败"

    return loaded

@tfmpl.figure_tensor
def draw_scatter(data, titles, colors):
    '''Draw scatter plots. One for each color.'''
    fig = tfmpl.create_figure(figsize=(8, 4))
    ax1 = fig.add_subplot(1, 2, 1)
    ax1.scatter(data[:, 0], data[:, 1], c=colors[0])
    ax1.set_title(titles[0])
    ax2 = fig.add_subplot(1, 2, 2)
    ax2.scatter(data[:, 0], data[:, 2], c=colors[1])
    ax2.set_title(titles[1])
    fig.tight_layout()
    return fig

def main(sample_index = None):
    model = load_model(save_path)
    y = model.inference(x)
    pts = tf.concat([x, y_d, y], axis=-1)

    summary_writer = tf.summary.create_file_writer(log_dir)
    with summary_writer.as_default():
        image_tensor = draw_scatter(pts, ["Ground-truth Model", "Learned Model"], ['b',
'r'])
        image_summary = tf.summary.image("images", image_tensor, step=0)
    summary_writer.close()

if __name__ == "__main__":
    main()
```

为方便起见，文件 lr_test.py 附后：

lr_test.py

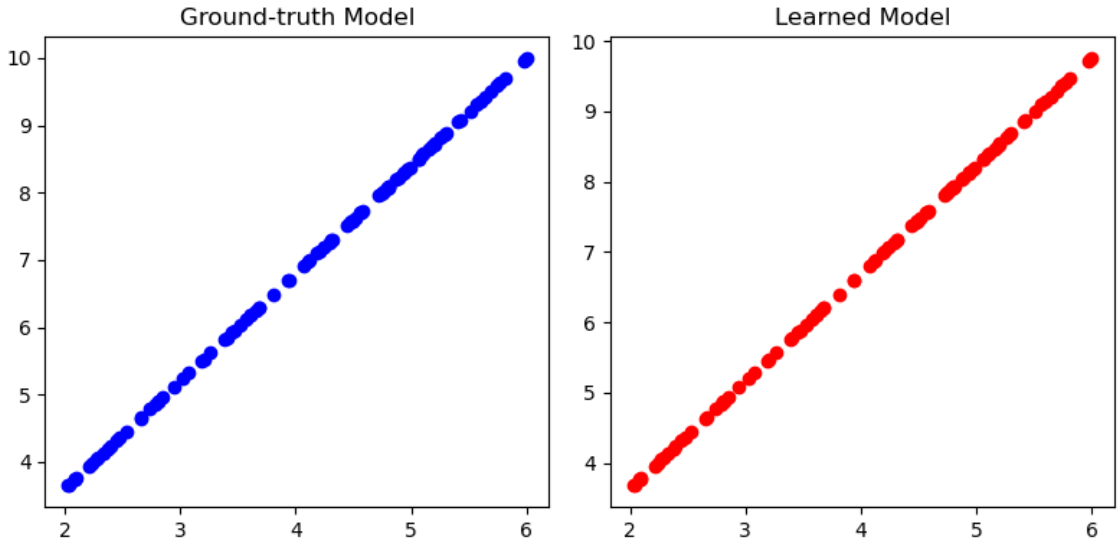
3. 在命令行执行脚本文件： python lr_test.py

4. 如果程序运行无误，则通过在命令行中执行如下命令启用 TensorBoard：

tensorboard --logdir=output-test

可以通过在浏览器中键入如下地址观察输出：

http://localhost:6006



小结：

从数据在真实模型的输出与训练模型的输出上的图示上可以看出，所建立的模型经过训练之后，与真实模型是有很高的拟合度的。

实作：

依据示例代码，测试所训练的非线性函数的模型的表现，生成数据的区间选择为$[-6, 6]$。请附修改后的代码，以及在浏览器中观察到的图像。