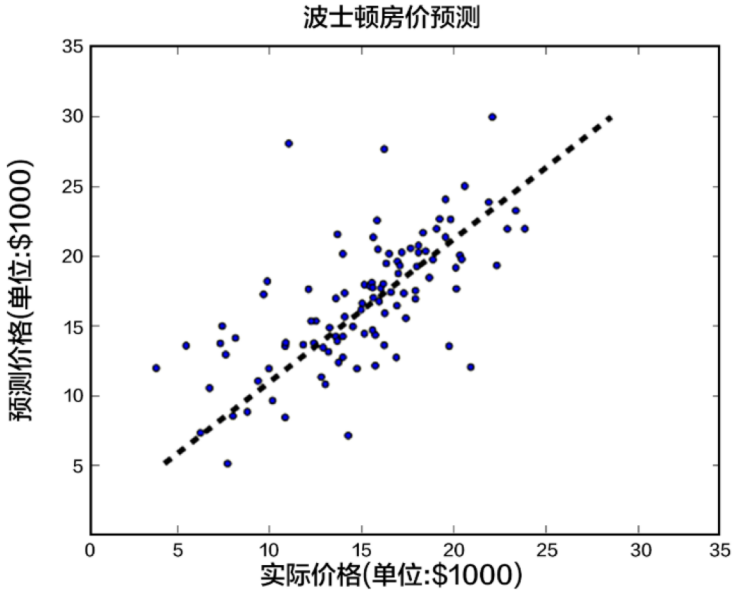
波士顿房价预测

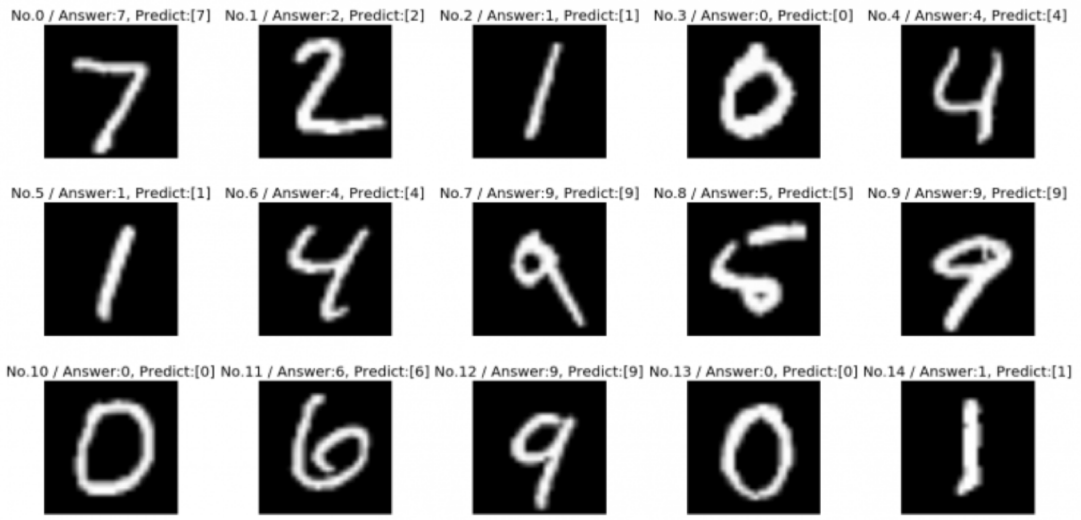
任务介绍

为帮助广大开发者快速入门，**飞桨PaddlePaddle官网**提供了两个经典例子：

房价预测



手写数字识别

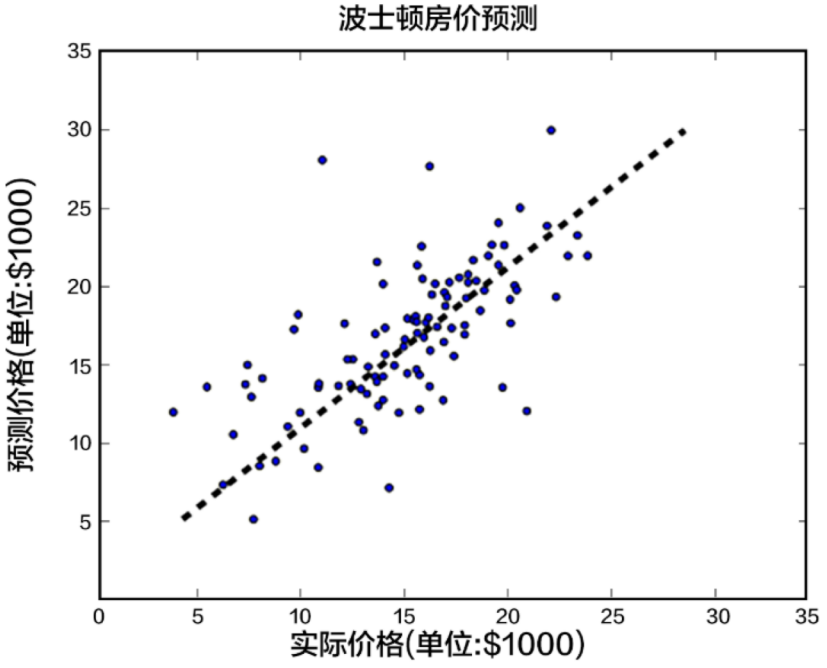


本教程将在AI Studio上实践基于线性回归模型的房价预测，帮助大家快速入门。

模型选择

经典的**线性回归**（Linear Regression ）模型主要用来预测一些存在着线性关系的数据集。本教程使用真实的数据集建立起一个房价预测模型，并且了解到机器学习中的若干重要概念。

我们使用从UCI Housing Data Set获得的波士顿房价数据集进行模型的训练和预测。**散点图展示了使用模型对部分房屋价格进行的预测**。其中，每个点的横坐标表示同一类房屋真实价格的中位数，纵坐标表示线性回归模型根据特征预测的结果，当二者值完全相等的时候就会落在虚线上。所以模型预测得越准确，则点离虚线越近。



数据集

◆波士顿房价数据集

- 数据集共506行
- 每行14列
  - ✓ 前13列用来描述房屋的各种信息
  - ✓ 最后一列为该类房屋价格中位数
- 飞桨PaddlePaddle提供自动加载模块
  - ✓ 训练集：

    paddle.dataset.uci_housing.train()
  - ✓ 测试集：

    paddle.dataset.uci_housing.test()

属性名	解释	类型
CRIM	该镇的人均犯罪率	连续值
ZN	占地面积超过25,000平方呎的住宅用地比例	连续值
INDUS	非零售商业用地比例	连续值
CHAS	是否邻近 Charles River	离散值，1=邻近；0=不邻近
NOX	一氧化氮浓度	连续值
RM	每栋房屋的平均客房数	连续值
AGE	1940年之前建成的自用单位比例	连续值
DIS	到波士顿5个就业中心的加权距离	连续值
RAD	到径向公路的可达性指数	连续值
TAX	全值财产税率	连续值
PTRATIO	学生与教师的比例	连续值
B	1000(BK - 0.63)^2，其中BK为黑人占比	连续值
LSTAT	低收入人群占比	连续值
MEDV	同类房屋价格的中位数	连续值

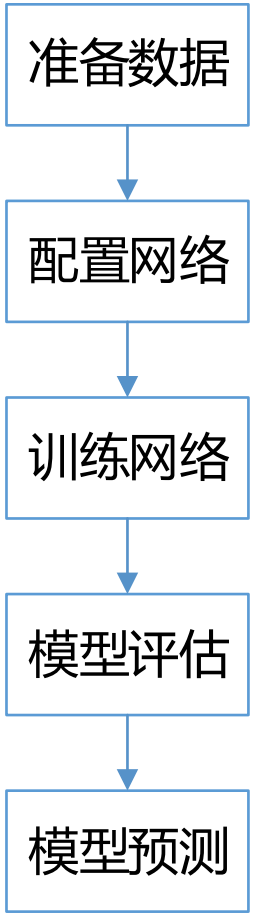
实践流程

- 实践流程
  - 准备数据
  - 配置网络
    - ✓ 定义网络
    - ✓ 定义损失函数
    - ✓ 定义优化算法
  - 训练网络
  - 模型评估
  - 模型预测

训练网络

定义好模型结构之后，我们要通过以下几个步骤进行模型训练。

1. 网络正向传播计算网络输出和损失函数。
2. 根据损失函数进行反向误差传播，将网络误差从输出层依次向前传递，并更新网络中的参数。
3. 重复1~2步骤，直至网络训练误差达到规定的程度或训练轮次达到设定值。



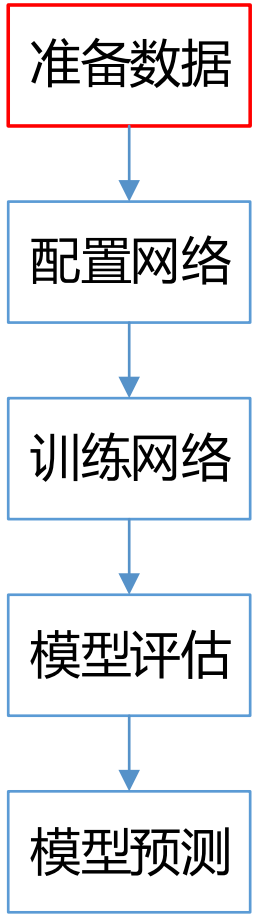
导入包

飞桨

◆ 导入必要的包

```
import paddle.fluid as fluid
import paddle
import numpy as np
import os
```

- paddle.fluid-→ 飞桨核心框架
- numpy------→ python基本库，用于科学计算
- os----------→ python的模块，可使用该模块对操作系统进行操作



准备数据

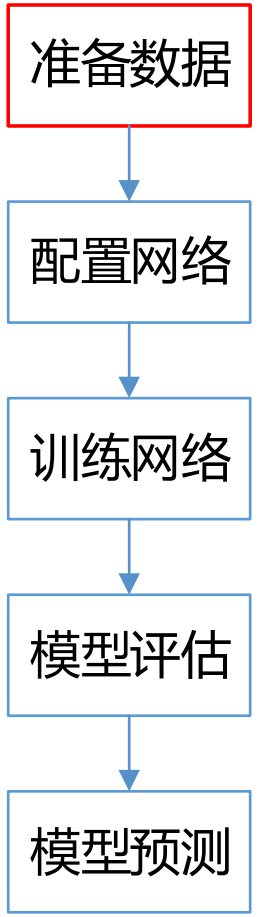
飞桨

◆ 训练数据集准备

```
BUF_SIZE=500
BATCH_SIZE=20

#用于训练的数据提供器，每次从缓存中随机读取批次大小的数据
train_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.uci_housing.train(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)
```

- ✓ paddle.dataset.uci_housing.train()表示获取uci_housing的训练集
- ✓ paddle.reader.shuffle()表示每次缓存BUF_SIZE个数据项，并进行打乱
- ✓ paddle.batch()表示按批次读取乱序后的数据，批次大小为BATCH_SIZE



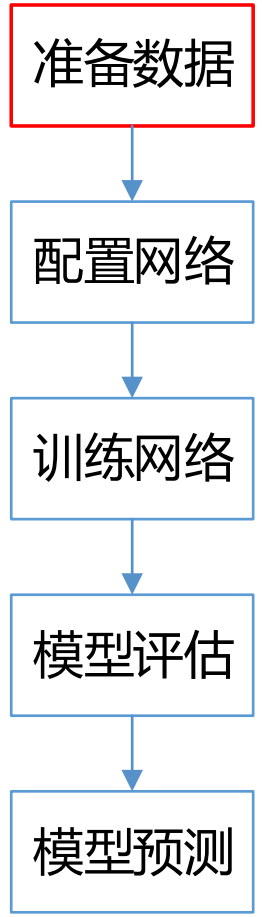
准备数据

飞桨

◆ 测试数据集准备

```
#用于测试的数据提供器，每次从缓存中随机读取批次大小的数据
test_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.uci_housing.test(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)
```

- ✓ paddle.dataset.uci_housing.test()表示获取uci_housing的测试集
- ✓ paddle.reader.shuffle()表示每次缓存BUF_SIZE个数据项，并进行打乱
- ✓ paddle.batch()表示按批次读取乱序后的数据，批次大小为BATCH_SIZE



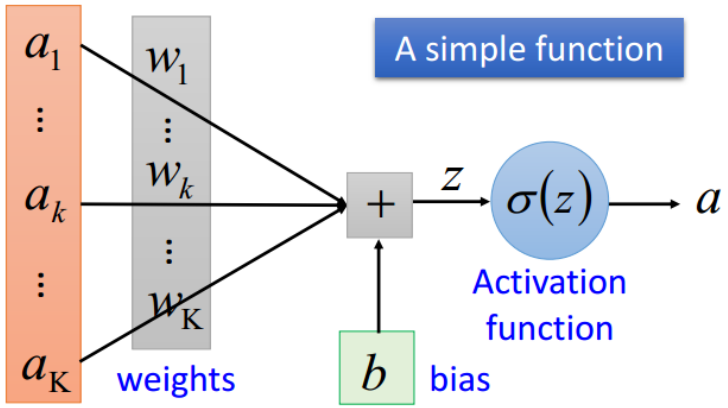
配置网络

- 定义简单的线性网络

对于波士顿房价数据集，假设属性和房价之间的关系可以被属性间的线性组合描述

$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$

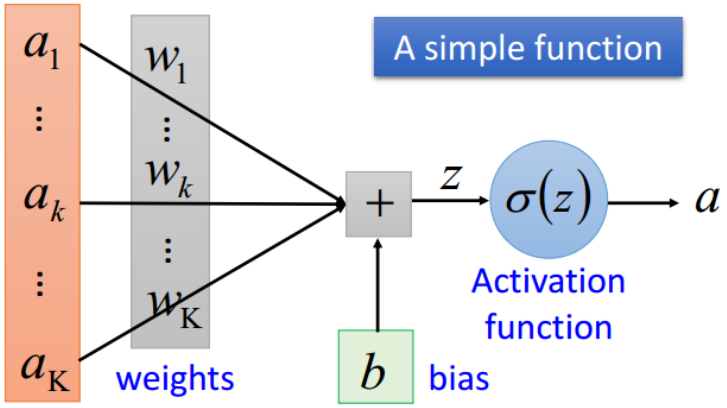
线性回归的本质就是一个采用线性激活函数的全连接层（fully-connected layer）



准备数据
配置网络
训练网络
模型评估
模型预测

配置网络

- 定义简单的线性网络




```
#定义张量变量x，表示13维的特征值
x = fluid.layers.data(name='x', shape=[13], dtype='float32')
```

配置数据层，定义x:名称为x,形状为[13],数据类型为float32

```
#定义张量y,表示目标值
y = fluid.layers.data(name='y', shape=[1], dtype='float32')
```

定义真实值y,名称为y,形状为[1],数据类型为float32

```
#定义一个简单的线性网络,连接输入和输出的全连接层
y_predict=fluid.layers.fc(input=x,size=1,act=None)
```

fluid.layers.fc():全连接层。定义一个一层的网络

- 准备数据
- 配置网络
- 训练网络
- 模型评估
- 模型预测

配置网络

飞桨

◆ 定义损失函数

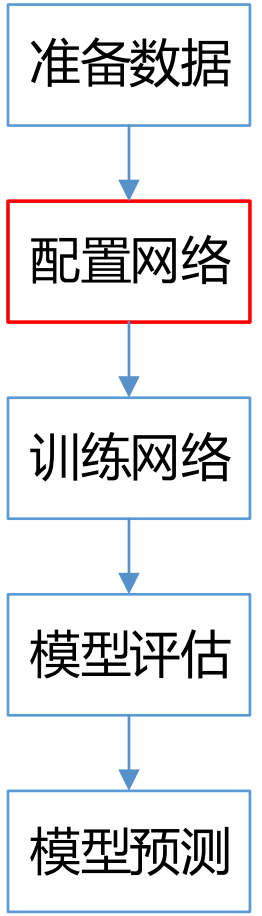
对于线性模型来讲，最常用的损失函数就是均方误差（Mean Squared Error, MSE）。

```
cost = fluid.layers.square_error_cost(input=y_predict, label=y) #求一个batch的损失值
avg_cost = fluid.layers.mean(cost)                              #对损失值求平均值
```

损失函数

损失函数定义了拟合结果和真实结果之间的差异，作为优化的目标直接关系模型训练的好坏。

Paddle fluid中提供了面向多种任务的多种类型的损失函数，如回归问题中的平均误差损失、分类问题中的交叉熵损失函数等



配置网络

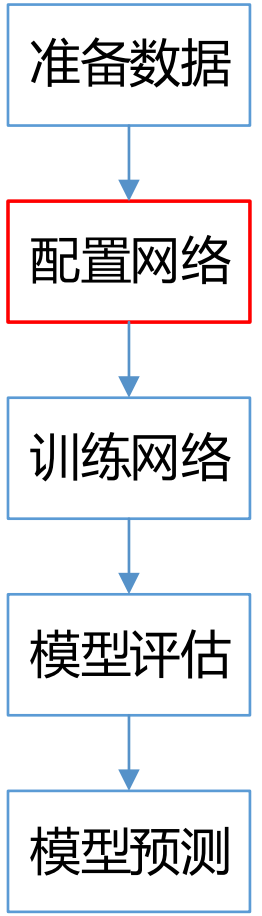
定义优化算法

使用随机梯度下降算法进行优化，learning_rate代表学习率

```
optimizer = fluid.optimizer.SGDOptimizer(learning_rate=0.001)
opts = optimizer.minimize(avg_cost)
```

随机梯度下降算法

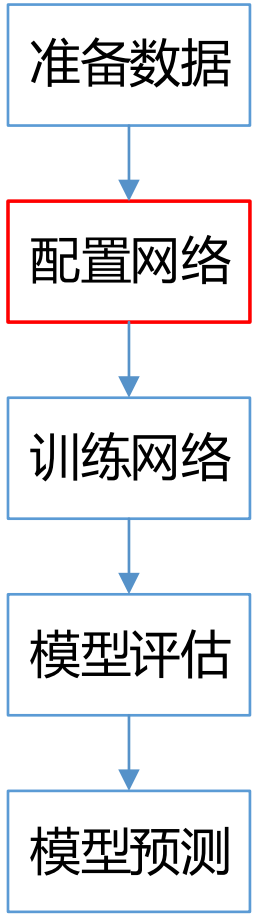
SGD算法是从样本中随机抽出一组，训练后按梯度更新一次，然后再抽取一组，再更新一次，在样本量及其大的情况下，可能不用训练完所有的样本就可以获得一个损失值在可接受范围之内的模型了。



配置网络

上述模型配置完毕后，得到两个fluid.Program

- **fluid.default_startup_program() :**
  - ✓ 参数初始化操作会被写入fluid.default_startup_program()
- **fluid.default_main_program():**
  - ✓ 用于获取默认或全局main program(主程序)。
  - ✓ 该主程序用于训练和测试模型。fluid.layers 中的所有layer函数可以向 default_main_program 中添加算子和变量。
  - ✓ 是Fluid许多编程接口的缺省值。



训练网络

◆ 创建训练用Executor

```
use_cuda = False
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())
```

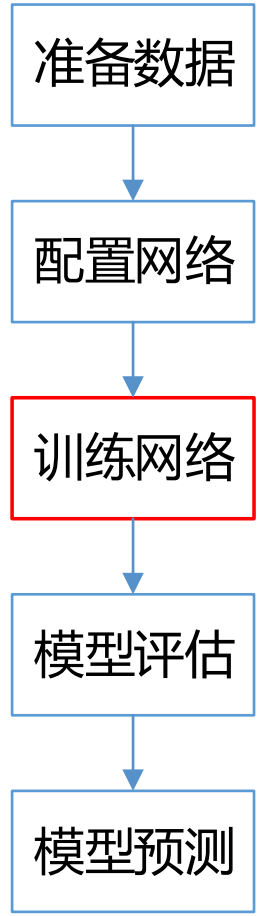
✓ 指定程序运行的设备，fluid.CPUPlace()和 fluid.CUDAPlace(0)分别表示为CPU和GPU

✓ 创建一个Executor实例

✓ Executor接收传入的Program，并通过run()方法运行Program

◆ 定义数据映射器

```
# 定义输入数据维度
feeder = fluid.DataFeeder(place=place, feed_list=[x, y])
```



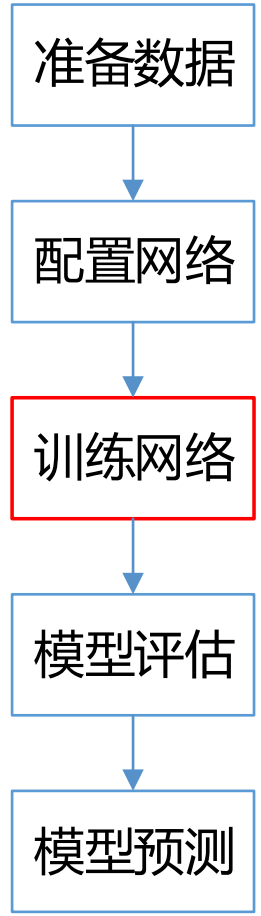
训练网络

飞桨

◆ 开始进行网络训练

```
EPOCH_NUM=100
model_save_dir = "/home/aistudio/data/fit_a_line.inference.model"

for pass_id in range(EPOCH_NUM):                                    #训练EPOCH_NUM轮
    # 开始训练并输出最后一个batch的损失值
    train_cost = 0
    for batch_id, data in enumerate(train_reader()):                #遍历train_reader迭代器
        train_cost = exe.run(program=fluid.default_main_program(),#运行主程序
                             feed=feeder.feed(data),                #喂入一个batch的训练数据
                             fetch_list=[avg_cost])
    print("Pass:%d, Cost:%0.5f" % (pass_id, train_cost[0][0]))      #打印最后一个batch的损失
```



Executor接收传入的Program,并根据feed map(输入映射表)和fetch_list(结果获取表) 向Program中添加feed operators(数据输入算子)和fetch operators（结果获取算子)。 feed map为该Program提供输入数据。fetch_list提供Program训练结束后用户预期的变量。

训练网络

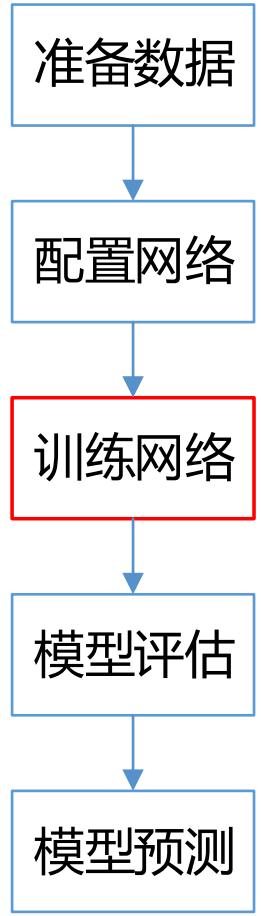
飞桨

◆ 开始进行网络训练

```
test_program = fluid.default_main_program().clone(for_test=True)

# 开始测试并输出最后一个batch的损失值
test_cost = 0
for batch_id, data in enumerate(test_reader()):
    test_cost= exe.run(program=test_program,
                       feed=feeder.feed(data),
                       fetch_list=[avg_cost])
print('Test:%d, Cost:%0.5f' % (pass_id, test_cost[0][0]))
```

```
#遍历test_reader迭代器
#运行测试程序
#喂入一个batch的测试数据
#fetch均方误差
#打印最后一个batch的损失值
```



Executor接收传入的Program,并根据feed map(输入映射表)和fetch_list(结果获取表) 向Program中添加feed operators(数据输入算子)和fetch operators（结果获取算子)。 feed map为该Program提供输入数据。fetch_list提供Program训练结束后用户预期的变量。

训练网络

◆ 保存训练模型

```
#保存模型
# 如果保存路径不存在就创建
if not os.path.exists(model_save_dir):
    os.makedirs(model_save_dir)
print ('save models to %s' % (model_save_dir))
#保存训练参数到指定路径中，构建一个专门用预测的program
fluid.io.save_inference_model(model_save_dir,    #保存推理model的路径
                              ['x'],             #推理（inference）需要 feed 的数据
                              [y_predict],       #保存推理（inference）结果的 Variables
                              exe)               #exe 保存 inference model
```

save_inference_model()

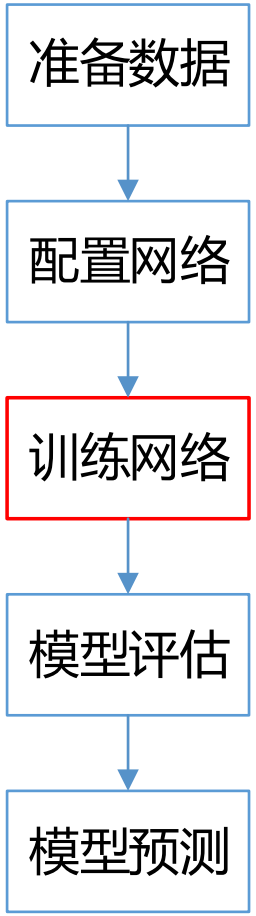
构建一个专门用于推理的 Program，然后 Executor 把它和所有相关参数保存到 dirname 中

第一个参数：**dirname (str)** – 保存推理model的路径

第二个参数：**feeded_var_names (list[str])** – 推理（inference）需要 feed 的数据

第三个参数：**target_vars (list[Variable])** – 保存推理（inference）结果的 Variables

第四个参数：**executor (Executor)** – executor 保存 inference model



模型评估

飞桨

- 网络训练过程中输出如下：

```
Pass:0, Cost:104.53711
Test:0, Cost:247.74060
Pass:1, Cost:73.07610
Test:1, Cost:197.55109
Pass:2, Cost:50.99372
Test:2, Cost:158.19765
Pass:3, Cost:35.97136
Test:3, Cost:127.46747
Pass:4, Cost:26.17634
Test:4, Cost:103.49704
Pass:5, Cost:20.20255
Test:5, Cost:84.81793
Pass:6, Cost:16.97699
Test:6, Cost:70.27448
Pass:7, Cost:15.68605
Test:7, Cost:58.95842
Pass:8, Cost:15.71751
Test:8, Cost:50.15700
Pass:9, Cost:16.61491
Test:9, Cost:43.31175
```

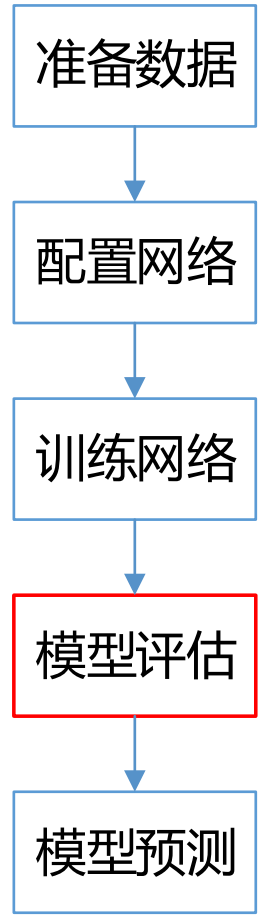
测试

测试与预测不同。

测试是为了试用于测试数据集预测并与真实结果对比，评估当前模型的好坏。

测试集不属于训练集，所以测试集的预测结果的好坏能够体现模型的泛化能力。

在训练过程中，我们可以看到输出的损失值在不断减小，证明我们的模型在不断收敛。



模型预测

飞桨

- 准备数据
- 配置网络
- 训练网络
- 模型评估
- 模型预测

◆ 创建预测用Executor

```
infer_exe = fluid.Executor(place)     #创建推测用的executor
inference_scope = fluid.core.Scope() #Scope指定作用域
```

◆ 读取预测模型

```
#从指定目录中加载 推理model(inference model)
[inference_program,                                    #推理的program
 feed_target_names,                                    #需要在推理program中提供数据的变量名称
 fetch_targets] = fluid.io.load_inference_model(#fetch_targets: 推断结果
                                 model_save_dir, #model_save_dir:模型训练路径
                                 infer_exe)      #infer_exe: 预测用executor
```

load_inference_model() 这个函数的返回有三个元素的元组

- Program 是一个 Program ，它是推理Program。
- feed_target_names 是一个str列表，它包含需要在推理 Program 中提供数据的变量的名称。
- fetch_targets`是一个 Variable 列表，从中我们可以得到推断结果。

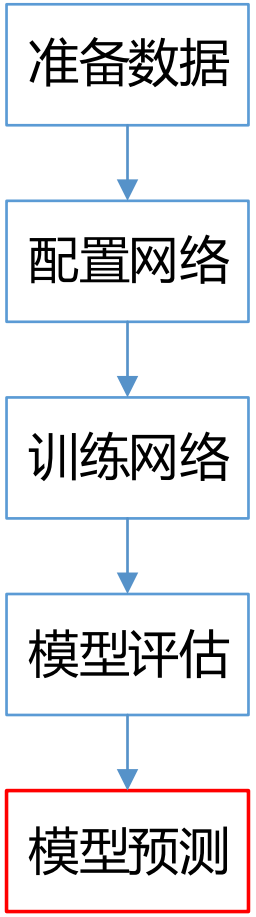
模型预测

飞桨

◆ 定义预测数据

```
#获取预测数据
infer_reader = paddle.batch(paddle.dataset.uci_housing.test(),
                            batch_size=10)
#从test_reader中分割x
test_data = next(infer_reader())
test_x = np.array([data[0] for data in test_data]).astype("float32")
test_y= np.array([data[1] for data in test_data]).astype("float32")
```

- ✓ paddle.dataset.uci_housing.test()表示获取uci_housing的测试集
- ✓ paddle.batch()表示按批次读取乱序后的数据，批次大小为10
- ✓ 将数据集分为test_x和test_y，将test_x喂入训练好的模型



模型预测

飞桨

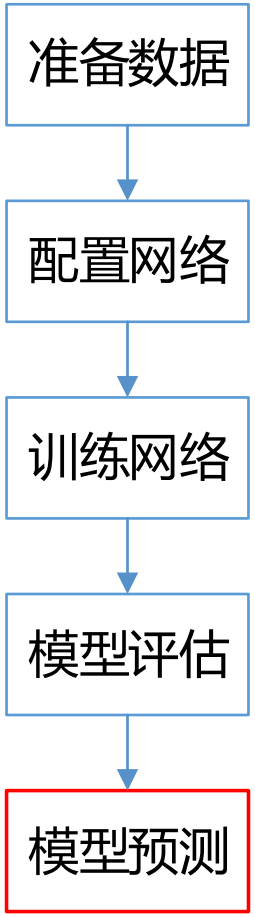
◆ 开始预测

```
results = infer_exe.run(inference_program,                                  #预测模型
                        feed={feed_target_names[0]: np.array(test_x)},      #喂入要预测的x值
                        fetch_list=fetch_targets)                           #得到推测结果
print("infer results: (House Price)")
for idx, val in enumerate(results[0]):
    print("%d: %.2f" % (idx, val))
print("ground truth:")
for idx, val in enumerate(test_y):
    print("%d: %.2f" % (idx, val))
```

输出结果

```
infer results: (House Price)
0: 14.61
1: 15.01
2: 14.33
3: 16.56
4: 14.92
5: 15.89
6: 15.64
7: 14.99
8: 12.00
9: 14.92
```

```
ground truth:
0: 8.50
1: 5.00
2: 11.90
3: 27.90
4: 17.20
5: 27.50
6: 15.00
7: 17.20
8: 17.90
9: 16.30
```



感谢聆听

有什么问题吗？