

深度学习

PaddlePaddle基础

DAY06

PaddlePaddle概述



PaddlePaddle简介

什么是PaddlePaddle

为什么要学PaddlePaddle

PaddlePaddle优点

PaddlePaddle缺点

国际竞赛获奖情况

行业应用

课程概览

学习资源

PaddlePaddle概述

PaddlePaddle简介

什么是PaddlePaddle

- PaddlePaddle (Parallel Distributed Deep Learning , 中文名飞桨)
是百度公司推出的开源、易学习、易使用的分布式深度学习平台
- 源于产业实践 , 在实际中有着优异表现
- 支持多种机器学习经典模型



为什么学习PaddlePaddle

- 开源、国产
- 能更好、更快解工程决实际问题



PaddlePaddle优点

- 易用性。语法简洁，API的设计干净清晰
- 丰富的模型库。借助于其丰富的模型库，可以非常容易的复现一些经典方法
- 全中文说明文档。首家完整支持中文文档的深度学习平台
- 运行速度快。充分利用 GPU 集群的性能，为分布式环境的并行计算进行加速



PaddlePaddle缺点

- 教材少
- 学习难度大、曲线陡峭



国际竞赛获奖情况

获奖模型 / 模块		国际竞赛	
视觉领域	PyramidBox模型	WIDER FACE三项测试子集	第一
	Attention Clusters网络模型	ActivityNet Kinetics Challenge 2017	第一
	StNet模型	ActivityNet Kinetics Challenge 2018	第一
	基于Faster R-CNN的多模型	Google AI Open Images-Object Detection Track	第一
增强学习框架PARL		NIPS AI for Prosthetics Challenge	第一



行业应用



农业

智能桃子分拣机
节约 90% 人力成本



林业

病虫害监测
识别准确率达到 90%



工业

公共场所控烟



零售

商品销量预测
单店生鲜报损降低 30%



人力

AI建立匹配系统
5倍面邀成功率



制造

智能零件分拣
人工效率增加 1 倍



石油

地震波藏油预测



通讯

基站网络故障预警



地产

智能楼宇管理
制冷系统节电 20%



汽车

充电桩故障预警
准确达 90%



➤ 官网

- ✓ 地址：<https://www.paddlepaddle.org.cn/>
- ✓ 内容：学习指南、文档、API手册

➤ 百度云智学院

- ✓ 地址：<http://abcxueyuan.cloud.baidu.com/#/courseDetail?id=14958>
- ✓ 内容：教学视频

➤ AIStudio

- ✓ 地址：<https://aistudio.baidu.com/aistudio/projectoverview/public/1>
- ✓ 内容：项目案例



体系结构



```
graph LR; A[体系结构] --> B[体系结构]; B --> C[总体架构]; B --> D[编译时与执行时]; B --> E[三个重要术语]; B --> F[案例1: 快速开始];
```

体系结构

体系结构

总体架构

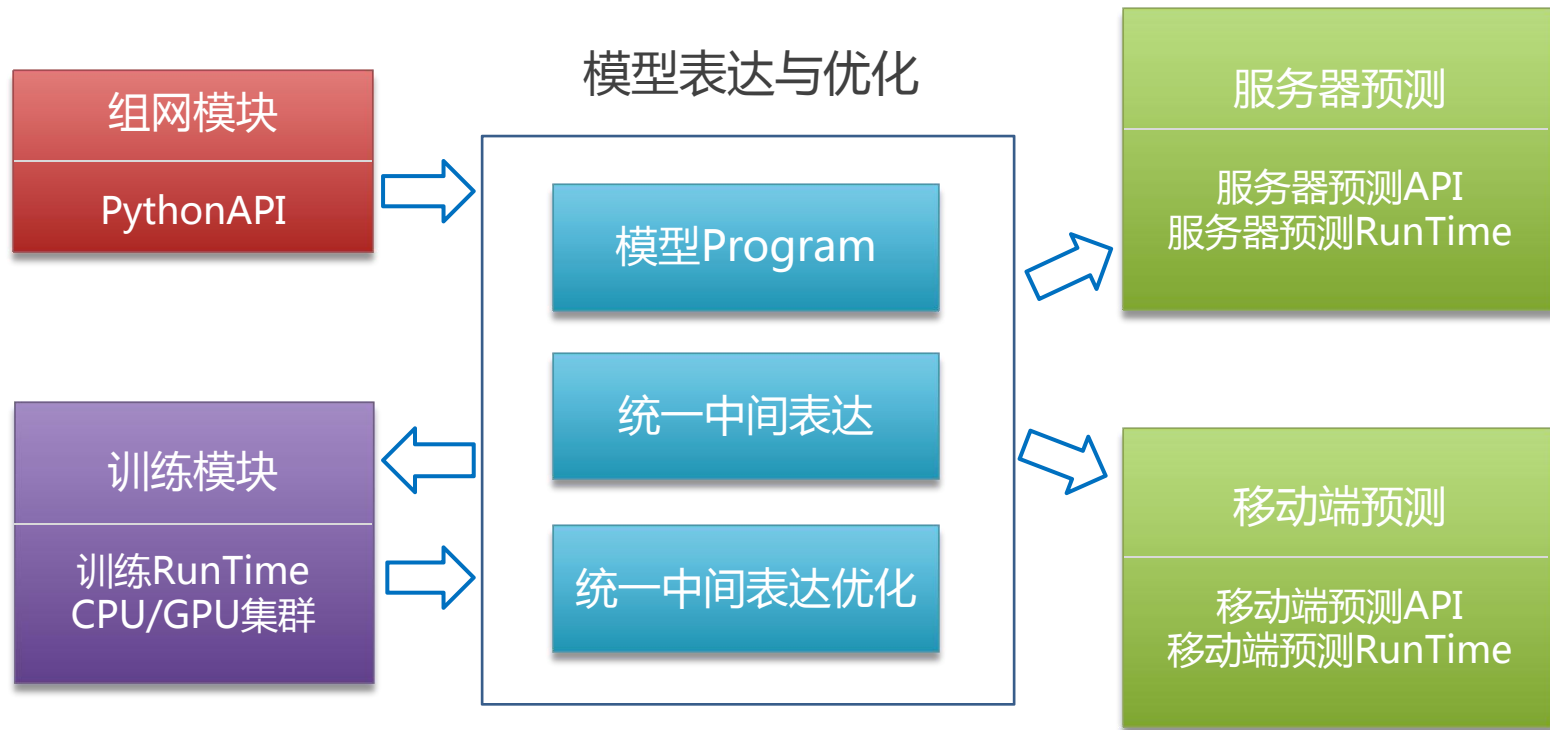
编译时与执行时

三个重要术语

案例1：快速开始

体系结构

总体架构

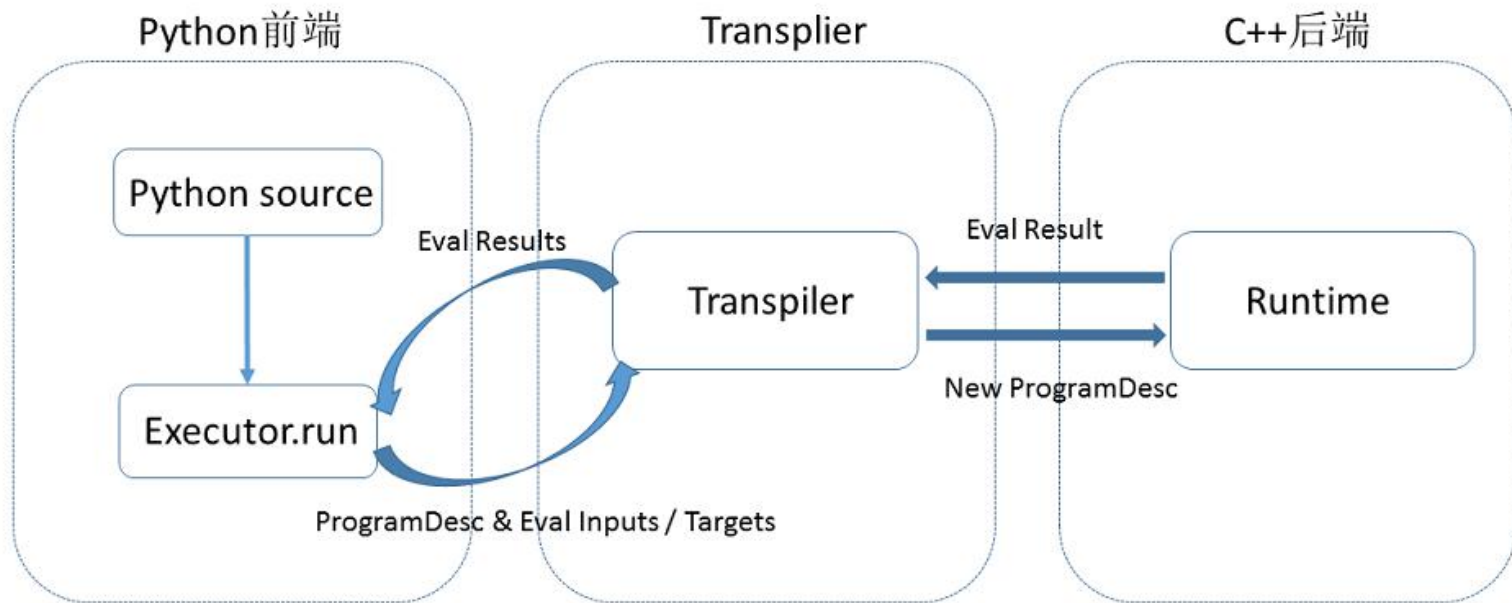


编译与执行过程

1. 用户编写的python程序通过调用 Paddle 提供的算子，向Program 中添加变量（ Tensor ）以及对变量的操作（ Operators 或者 Layers ）
2. 原始Program在框架内部转换为中间描述语言： ProgramDesc
3. Transpiler 接受一段 ProgramDesc ，输出一段变化后的 ProgramDesc ，作为后端 Executor 最终需要执行的 Program
4. 执行 ProgramDesc 中定义的 Operator （可以类比为程序语言中的指令），在执行过程中会为 Operator 创建所需的输入输出并进行管理



编译与执行过程（续）



三个重要术语

- Fluid : 定义程序执行流程
- Program : 对用户来说一个完整的程序
- Executor : 执行器 , 执行程序



案例1：快速开始

课堂练习

```
1 import paddle.fluid as fluid
2
3 # 创建两个类型为int64，形状为1行1列的张量
4 x = fluid.layers.fill_constant(shape=[1], dtype="int64", value=5)
5 y = fluid.layers.fill_constant(shape=[1], dtype="int64", value=1)
6 z = x + y
7 # print(z) # z为对象，此时还没有值
8
9 # 创建Executor执行器
10 place = fluid.CPUPlace() # 指定在CPU上运行
11 exe = fluid.Executor(place) # 创建执行器
12
13 result = exe.run(fluid.default_main_program(), fetch_list=[z])
14 print(result[0][0]) # result为1*1的张量
```



基本概念与操作

基本概念与操作

基本概念

张量

Layer

Variable

Program

Executor

Place

Optimizer

案例2：执行两个张量计算

实现线性回归

程序执行步骤

案例3：编写简单线性回归

Fluid API结构图

基本概念

张量

- 什么是张量

张量 (Tensor) : 多维数组或向量 , 同其它主流深度学习框架一样 , PaddlePaddle使用张量来承载数据

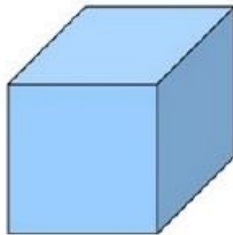
知识讲解



1d-tensor



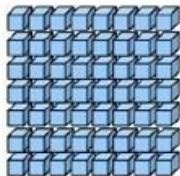
2d-tensor



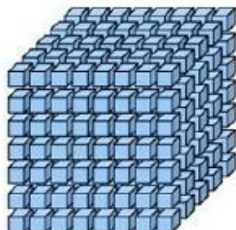
3d-tensor



4d-tensor



5d-tensor



6d-tensor



张量 (续1)

- 张量示例

灰度图像为二维张量 (矩阵) , 彩色图像为三维张量



0	1	239	780	1384	791	0	0	4	5	5	0	333	3432	0	0	632	632	
3	18	333	61	3832	3861	34	0	0	0	0	18	409	3832	1500	0	12	605	632
3	18	409	54	1239	980	72	77	52	92	40	322	2093	980	0	0	15	67	549
3	1	462	37	794	54	540	883	2300	3700	800	1409	2000	160	0	30	600	600	
46	1	332	361	467	82	85	900	379	2090	1400	173	3800	22	0	35	72	635	
46	1	332	361	466	71	72	97	147	134	203	3300	3700	22	0	17	79	605	
3	3	33	340	512	57	68	323	333	197	207	2000	370	0	0	30	79	605	
3	1	37	37	34	400	78	883	1400	147	203	225	335	0	0	30	79	605	
2	3	200	446	37	34	35	35	78	3740	234	1455	6000	36	2	21	78	605	
2	3	200	34	23	43	79	23	43	1359	205	93	323	70	0	23	78	72	
3	4	33	240	34	23	3932	379	330	3300	233	333	333	75	0	25	78	72	
46	5	33	23	20	20	37	330	384	900	1300	233	233	94	0	24	79	74	
16	5	35	25	30	30	33	345	340	900	133	252	252	74	4	20	79	75	
16	5	35	32	30	57	69	89	93	1200	120	252	252	154	20	20	80	75	
16	5	309	42	55	62	69	78	89	1000	1400	242	242	243	83	30	80	77	
22	3	3	209	20	55	64	69	80	78	1000	135	247	252	255	872	40	78	77
330	18	33	34	44	44	44	66	66	66	66	66	66	66	66	66	74	74	
330	18	34	37	40	63	65	134	134	134	134	245	252	252	255	132	66	75	



0	1	239	780	1384	791	0	0	4	5	5	0	333	3432	0	0	632	632	
3	18	333	61	3832	3861	34	0	0	0	0	18	409	3832	1500	0	12	605	632
3	18	409	54	1239	980	72	77	52	92	40	323	2093	980	0	0	15	67	549
3	1	462	37	794	54	540	883	2300	3700	800	1409	2000	160	0	30	600	600	
46	1	332	361	467	82	85	900	379	2090	1400	173	3800	22	0	35	72	635	
46	1	332	361	466	71	72	97	147	134	203	3300	3700	22	0	17	79	605	
3	3	33	340	512	57	68	323	333	197	207	2000	370	0	0	30	79	605	
3	1	37	37	34	400	78	883	1400	147	203	225	335	0	0	30	79	605	
2	3	200	446	37	34	35	35	78	3740	234	1455	6000	36	2	21	78	605	
2	3	200	34	23	43	79	23	43	1359	205	93	323	70	0	23	78	72	
3	4	33	240	34	23	3932	379	330	3300	233	333	333	75	0	25	78	72	
46	5	33	23	20	20	37	330	384	900	1300	233	233	94	0	24	79	74	
16	5	35	25	30	30	33	345	340	900	133	252	252	74	4	20	79	75	
16	5	35	32	30	57	69	89	93	1200	120	252	252	154	20	20	80	75	
16	5	309	42	55	62	69	78	89	1000	1400	242	242	243	83	30	80	77	
22	3	3	209	20	55	64	69	80	78	1000	135	247	252	255	872	40	78	77
330	18	33	34	44	44	44	66	66	66	66	66	66	66	66	66	74	74	
330	18	34	37	40	63	65	134	134	134	134	245	252	252	255	132	66	75	



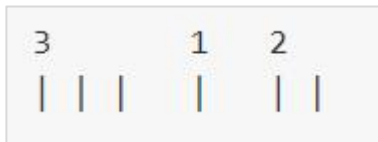
LoDTensor

- LoD(Level-of-Detail) Tensor是Paddle的高级特性，是对Tensor的一种扩充。LoDTensor通过牺牲灵活性来提升训练的效率。
- LoDTensor用来处理变长数据信息，将长度不一致的维度拼接为一个大的维度，并引入了一个索引数据结构（LoD）来将张量分割成序列。



LoDTensor (续)

- 假设一个mini-batch中有3个句子，每个句子中分别包含3个、1个和2个单词，我们可以用 $(3+1+2) \times D$ 维Tensor 加上一些索引信息来表示这个mini-batch：



- 假设存在一个mini-batch中包含3个句子、1个句子和2个句子的文章，每个句子都由不同数量的单词组成，则这个mini-batch的可以表示为2-Level的LoDTensor：



Layer

- 表示一个独立的计算逻辑，通常包含一个或多个operator（操作），如`layers.relu`表示ReLU计算；`layers.pool2d`表示pool操作。Layer的输入和输出为Variable。



Variable

- 表示一个变量，在paddle中，Variable 基本等价于 Tensor 。
Variable进入Layer计算，然后Layer返回Variable。创建变量方式：

```
x = fluid.layers.data(name="x", shape=[1], dtype="float32")  
y = fluid.layers.data(name="y", shape=[1], dtype="float32")
```



Python变量



Paddle变量



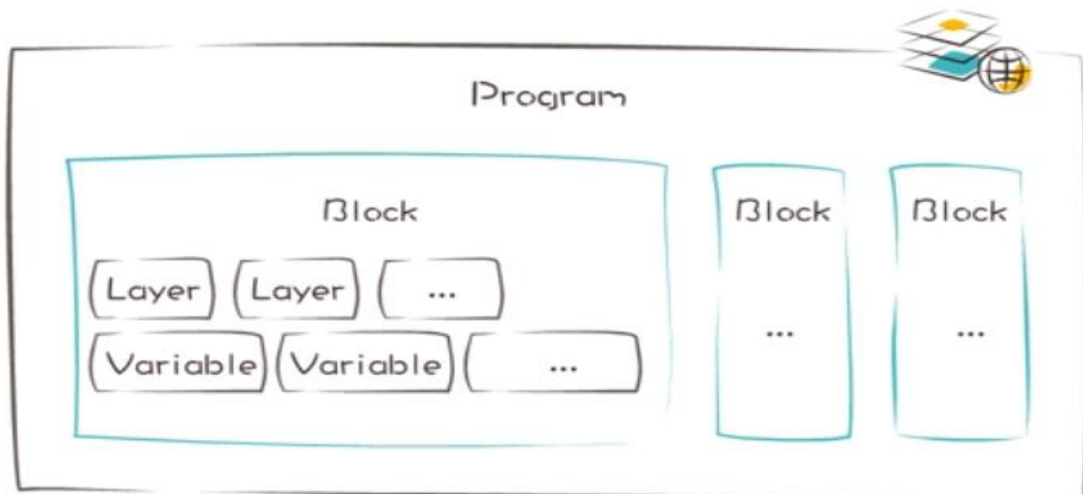
Variable（续）

- Paddle 中存在三种 Variable：
 - ✓ **模型中的可学习参数**：包括网络权重、偏置，生存期和整个训练任务一样长。通过 `fluid.layers.create_parameter` 来创建可学习参数
 - ✓ **占位 Variable**：Paddle 中使用 `fluid.data` 来接收输入数据，`fluid.data` 需要提供输入 Tensor 的形状信息，当遇到无法确定的维度时，相应维度指定为 `None`
 - ✓ **常量 Variable**：通过 `fluid.layers.fill_constant` 来实现常量 Variable



Program

- Program包含Variable定义的多个变量和Layer定义的多个计算，是一套完整的计算逻辑。从用户角度来看，Program是顺序、完整执行的。**program 的作用是存储网络结构，但不存储参数。**



Program（续）

- 用户完成网络定义后，一段 Paddle 程序中通常存在 2 个 Program
 - ✓ **fluid.default_startup_program**：定义了模型参数初始化、优化器参数初始化、reader初始化等各种操作。该program可以由框架自动生成，使用时无需显式地创建
 - ✓ **fluid.default_main_program**：定义了神经网络模型，前向反向计算，以及模型参数更新、优化器参数更新等各种操作



Scope

- scope 在 paddle 里可以看作变量空间，存储fluid创建的变量。变量存储于unordered_map 数据结构中，该结构类似于python中的dict，键是变量的名字，值是变量的指针。
- 一个 paddle 程序有一个默认的全局 scope（可以通过 fluid.global_scope() 获取）。如果没有主动创建 scope 并且通过 fluid.scope_guard() 替换当前 scope，那么所有参数都在全局 scope 中。**参数创建的时机不是在组网时，而是在 executor.run() 执行时。**
- program 和 scope 配合，才能表达完整模型（模型=网络结构+参数）



- Executor用来接收并执行Program，会一次执行Program中定义的所有计算。通过feed来传入参数，通过fetch_list来获取执行结果。

```
outs = exe.run(fluid.default_main_program(), # 默认程序上执行
               feed=params, # 喂入参数
               fetch_list=[result]) # 获取结果
```



Place

- PaddlePaddle可以运行在Intel CPU , Nvidia GPU , ARM CPU和更多嵌入式设备上 , 可以通过Place用来指定执行的设备 (CPU或GPU) 。

```
1 place = fluid.CPUPlace() # 指定CPU执行
2 place = fluid.CUDAPlace(0) # 指定GPU执行
```



Optimizer

- 优化器，用于优化网络，一般用来对损失函数做梯度下降优化，从而求得最小损失值

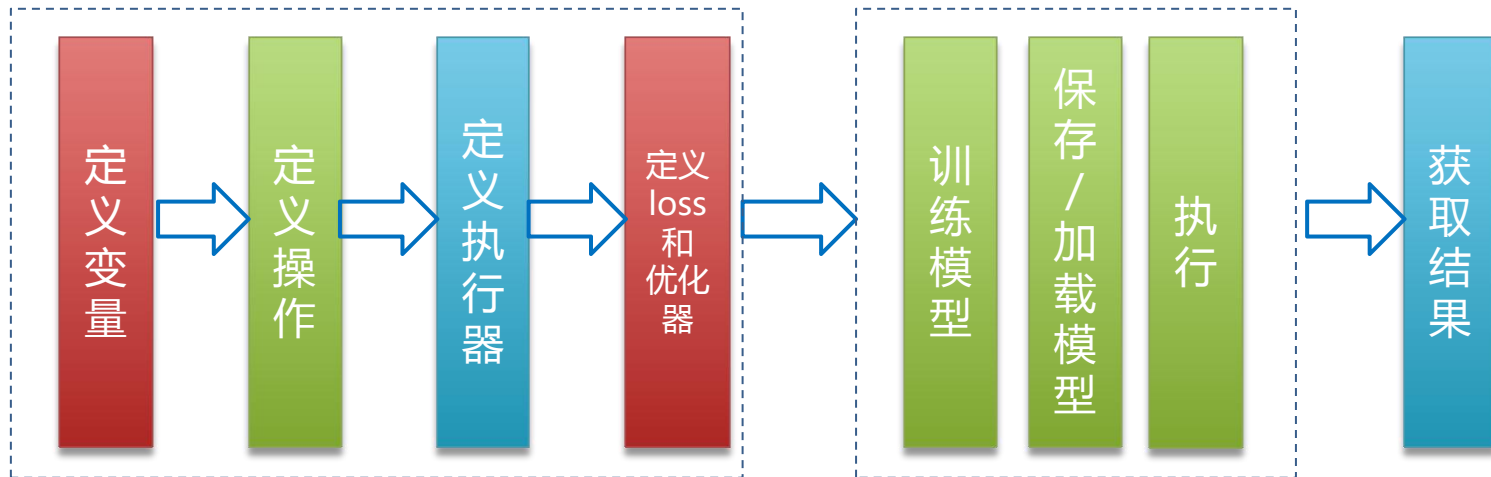


案例2：执行两个张量计算

```
1  import paddle.fluid as fluid
2  import numpy
3
4  # 创建x, y两个2行3列, 类型为float32的变量(张量)
5  x = fluid.layers.data(name="x", shape=[2, 3], dtype="float32")
6  y = fluid.layers.data(name="y", shape=[2, 3], dtype="float32")
7  x_add_y = fluid.layers.elementwise_add(x, y) # 两个张量按元素相加
8  x_mul_y = fluid.layers.elementwise_mul(x, y) # 两个张量按元素相乘
9
10 place = fluid.CPUPlace() # 指定在CPU上执行
11 exe = fluid.Executor(place) # 创建执行器
12 exe.run(fluid.default_startup_program()) # 初始化网络
13
14 a = numpy.array([[1, 2, 3],
15                 [4, 5, 6]]) # 输入x, 并转换为数组
16 b = numpy.array([[1, 1, 1],
17                 [2, 2, 2]]) # 输入y, 并转换为数组
18 params = {"x": a, "y": b}
19 outs = exe.run(fluid.default_main_program(), # 默认程序上执行
20               feed=params, # 喂入参数
21               fetch_list=[x_add_y, x_mul_y]) # 获取结果
22 for i in outs:
23     print(i)
```



程序执行步骤



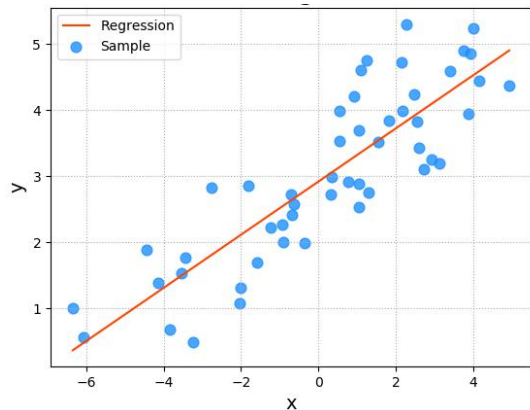
案例3：编写简单线性回归

➤ 任务：

- ✓ 给出输入样本
- ✓ 给出实际输出样本
- ✓ 找出 $y = wx + b$ 公式中的 w 和 b

➤ 思路：

- ✓ 定义输入数据、实际输出结果
- ✓ 将数据送入神经网络进行训练（全连接网络，即分类器）
- ✓ 根据实际输出、预测输出之间的损失值，进行梯度下降，直到收敛到极小值为止



案例3：编写简单线性回归（续）

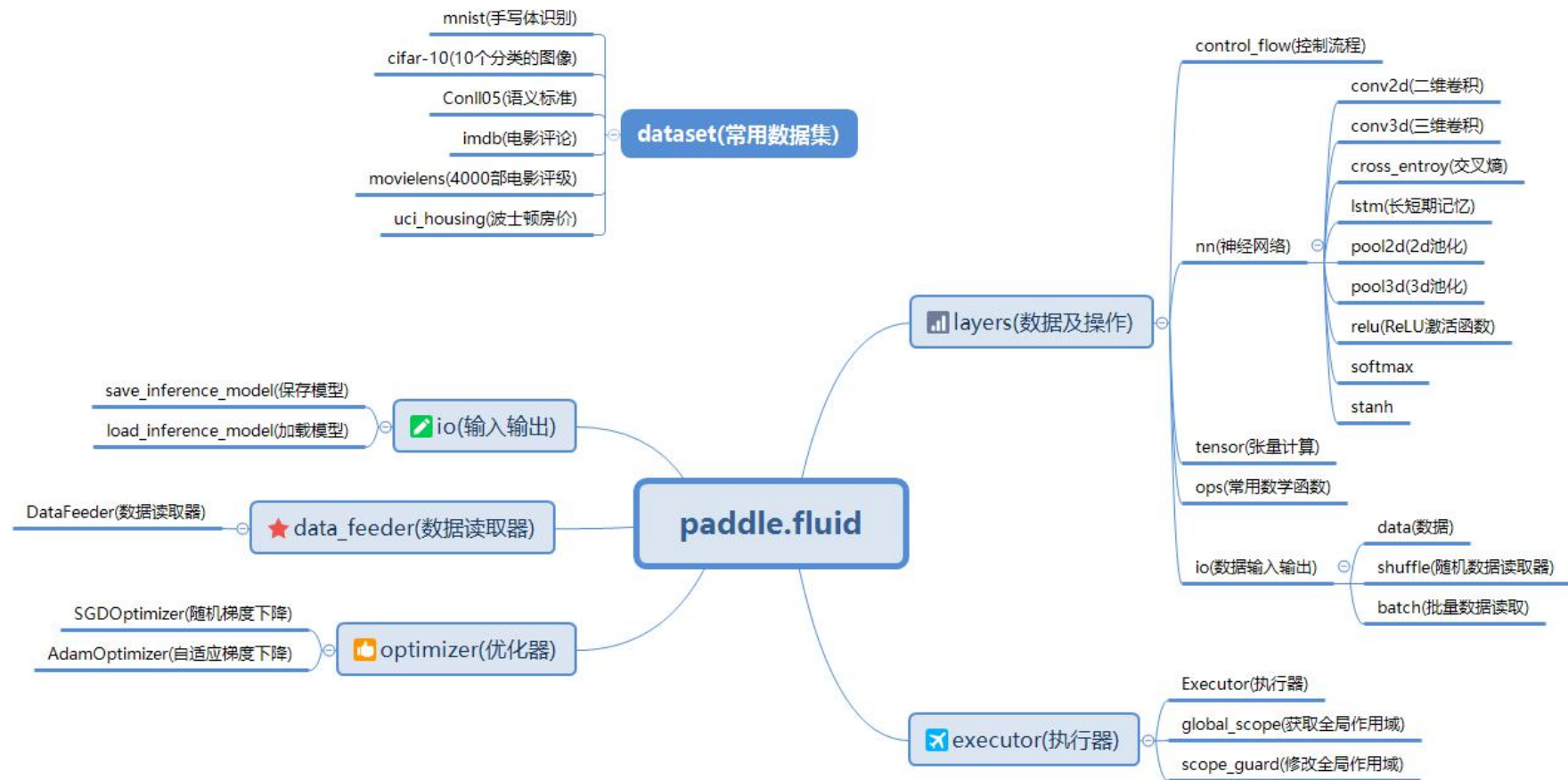
➤ 技术要点：

- ✓ 神经网络，选择 `fluid.layers.fc()`，该函数在神经网络中建立一个全连接层。接收多个输入，为每个输入分配一个权重 w ，并维护一个偏置值 b ；预测时产生一个输出
- ✓ 损失函数：回归问题，选择均方差 `fluid.layers.square_error_cost`和 `fluid.layers.mean()`作为损失函数
- ✓ 优化器：随机梯度下降优化器 `fluid.SGD`，做梯度下降计算

代码见：simple_lr.py



fluid API 结构图



数据准备

数据准备

数据准备

什么是数据准备

为什么需要数据准备

案例4：使用reader

实现多元回归

数据集及任务

思路

执行结果

案例5：波士顿房价预测

数据准备

深度学习数据读取要求

- **从文件读入数据。** 因为程序无法保存大量数据，数据一般保存到文件中，所以需要单独的数据读取操作
- **批量快速读入。** 深度学习样本数据量较大，需要快速、高效读取（批量读取模式）
- **随机读入。** 为了提高模型泛化能力，有时需要随机读取数据（随机读取模式）



案例4：使用reader

- 自定义reader creator，从文本文件test.txt中读取一行数据

```
1 import numpy
2 import paddle
3
4 def reader_creator(file_path):
5     def reader():
6         with open(file_path, "r") as f:
7             lines = f.readlines()
8             for line in lines:
9                 yield line
10    return reader
11
12 reader = reader_creator("test.txt")
13 for data in reader():
14     print(data, end="")
```



案例4：使用reader（续1）

- 从上一个reader中以随机方式读取数据

```
1 import numpy
2 import paddle
3
4 def reader_creator(file_path):
5     def reader():
6         with open(file_path, "r") as f:
7             lines = f.readlines()
8             for line in lines:
9                 yield line
10    return reader
11
12 reader = reader_creator("test.txt")
13 shuffle_reader = paddle.reader.shuffle(reader, 10)
14 for data in shuffle_reader():
15     print(data, end="")
```



案例4：使用reader（续2）

- 从上一个随机读取器中，分批次读取数据

```
1 import numpy
2 import paddle
3
4 def reader_creator(file_path):
5     def reader():
6         with open(file_path, "r") as f:
7             lines = f.readlines()
8             for line in lines:
9                 yield line
10    return reader
11
12 reader = reader_creator("test.txt")
13 shuffle_reader = paddle.reader.shuffle(reader, 10)
14 batch_reader = paddle.batch(shuffle_reader, 3)
15 for data in batch_reader():
16     print(data, end="")
```



实现多元回归

数据集及任务

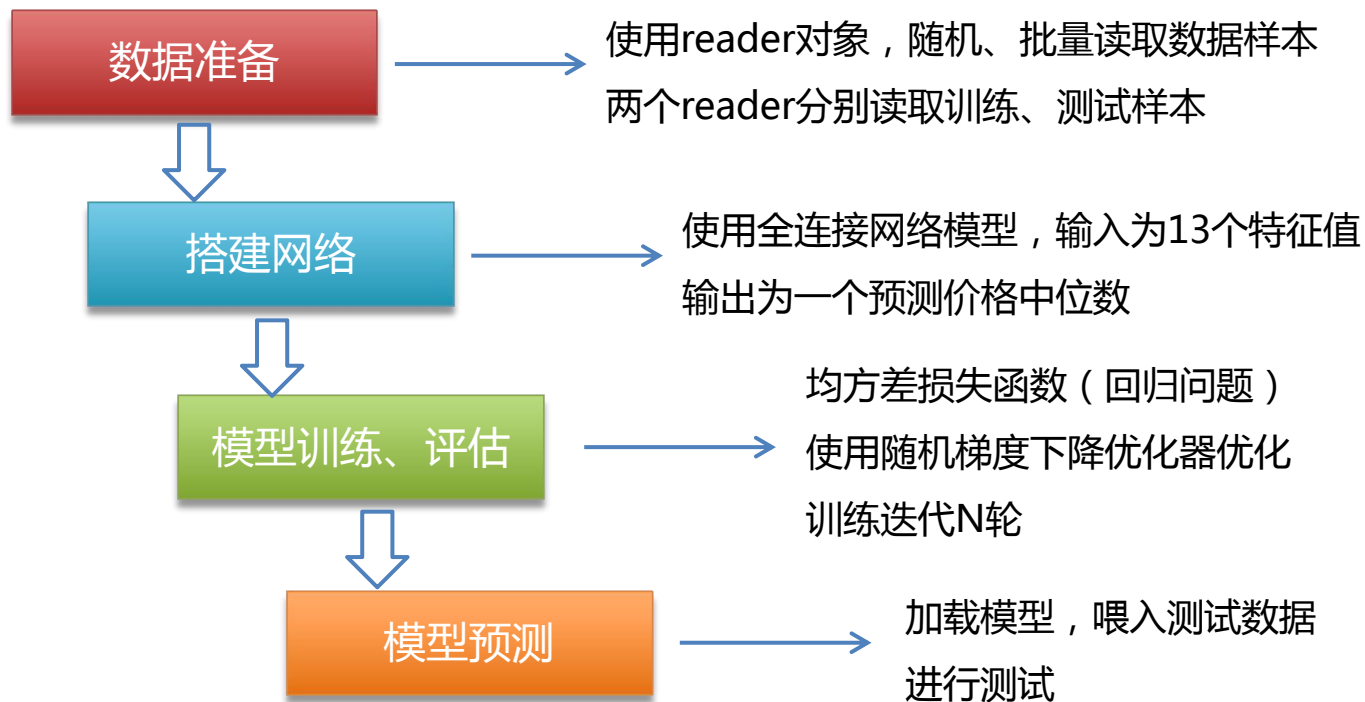
➤ 数据集介绍

- ✓ 数据量：506笔
- ✓ 特征数量：13个（见右图）
- ✓ 标签：价格中位数

➤ 任务：根据样本数据，预测房价中位数（回归问题）

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	$1000(BK-0.63)^2$	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

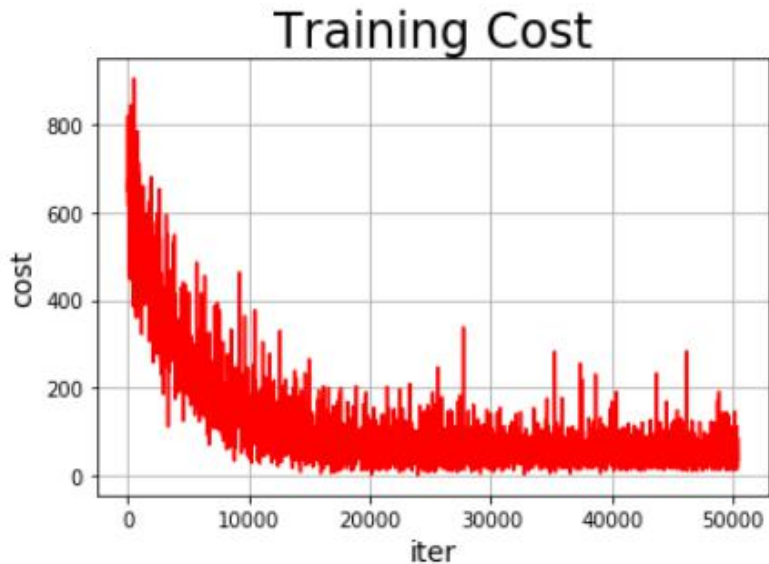




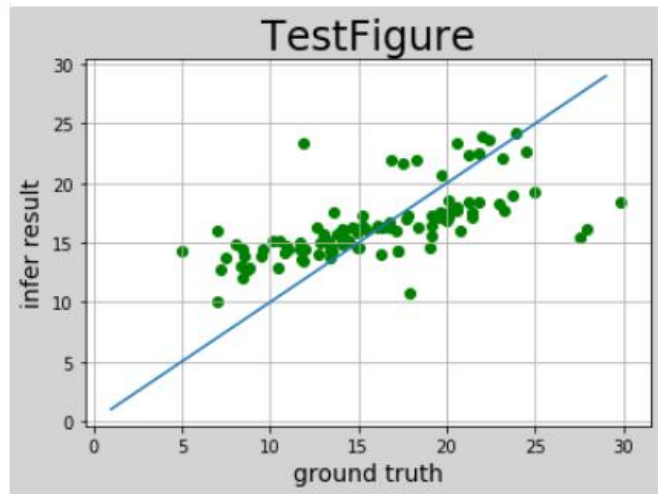
代码见：linear_regression.py



执行结果



损失函数收敛过程



预测值与实际值对比



案例5：波士顿房价预测

- 全部代码见：`uci_housing.py`



今日总结

- PaddlePaddle体系结构与基本概念
 - Tensor, Layer, Program, Variable, Executor, Place
 - Fluid API组织结构
- 案例：
 - 简单线性回归
 - 机器学习经典案例：波士顿房价预测