Tedu.cn
达内教育

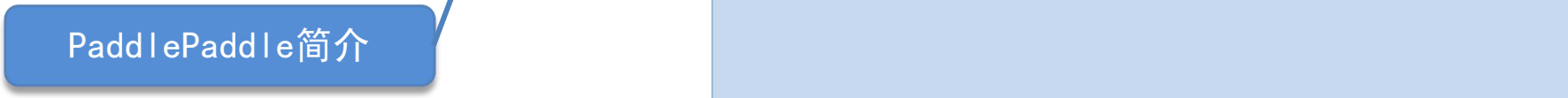
深度学习

PaddlePaddle基础

DAY03

PaddlePaddle

PaddlePaddle简介



- PaddlePaddle简介
  - 什么是PaddlePaddle
  - 为什么要学PaddlePaddle
  - PaddlePaddle优缺点
  - 课程内容预览

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

PaddlePaddle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

PaddlePaddle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

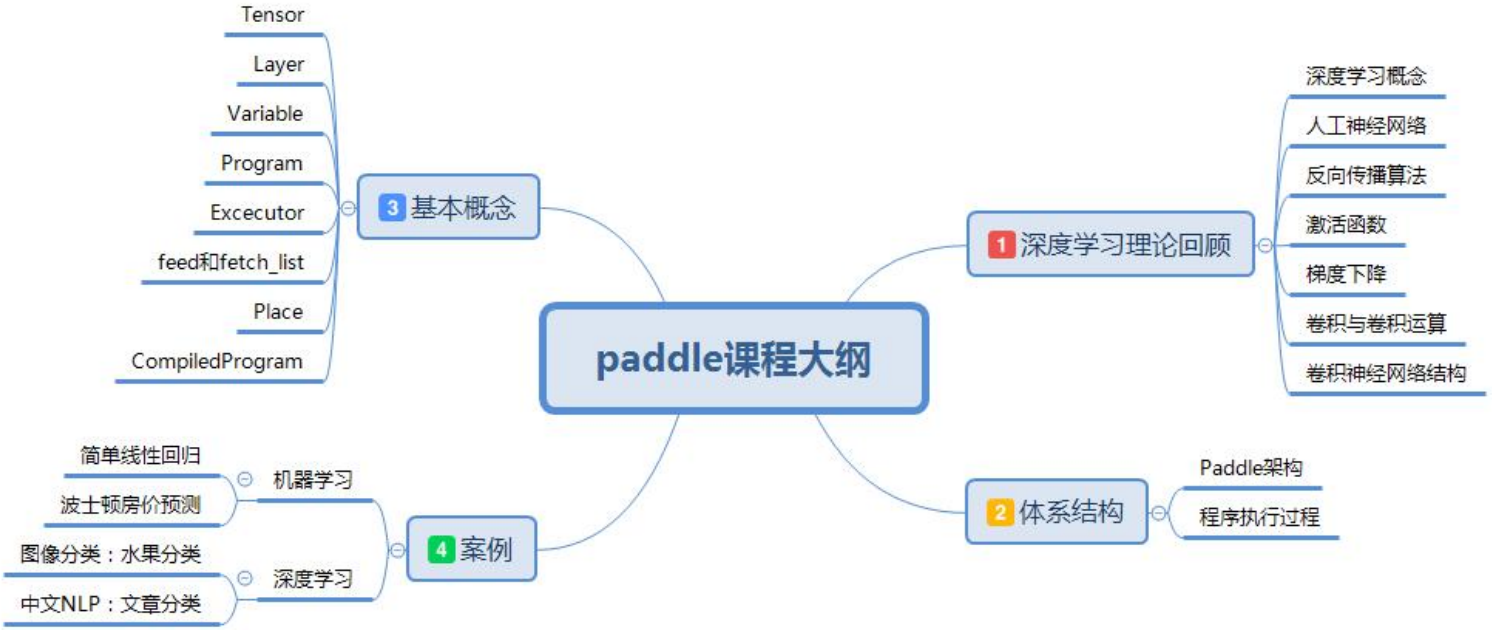
地产

智能楼宇管理
制冷系统节电 20%

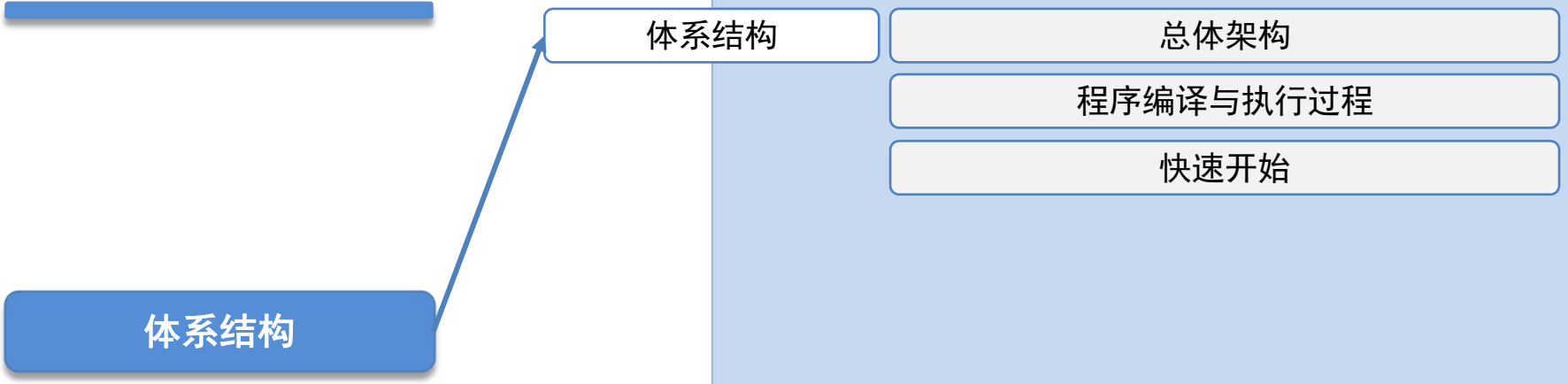
汽车

充电桩故障预警
准确达 90%

PaddlePaddle课程概览

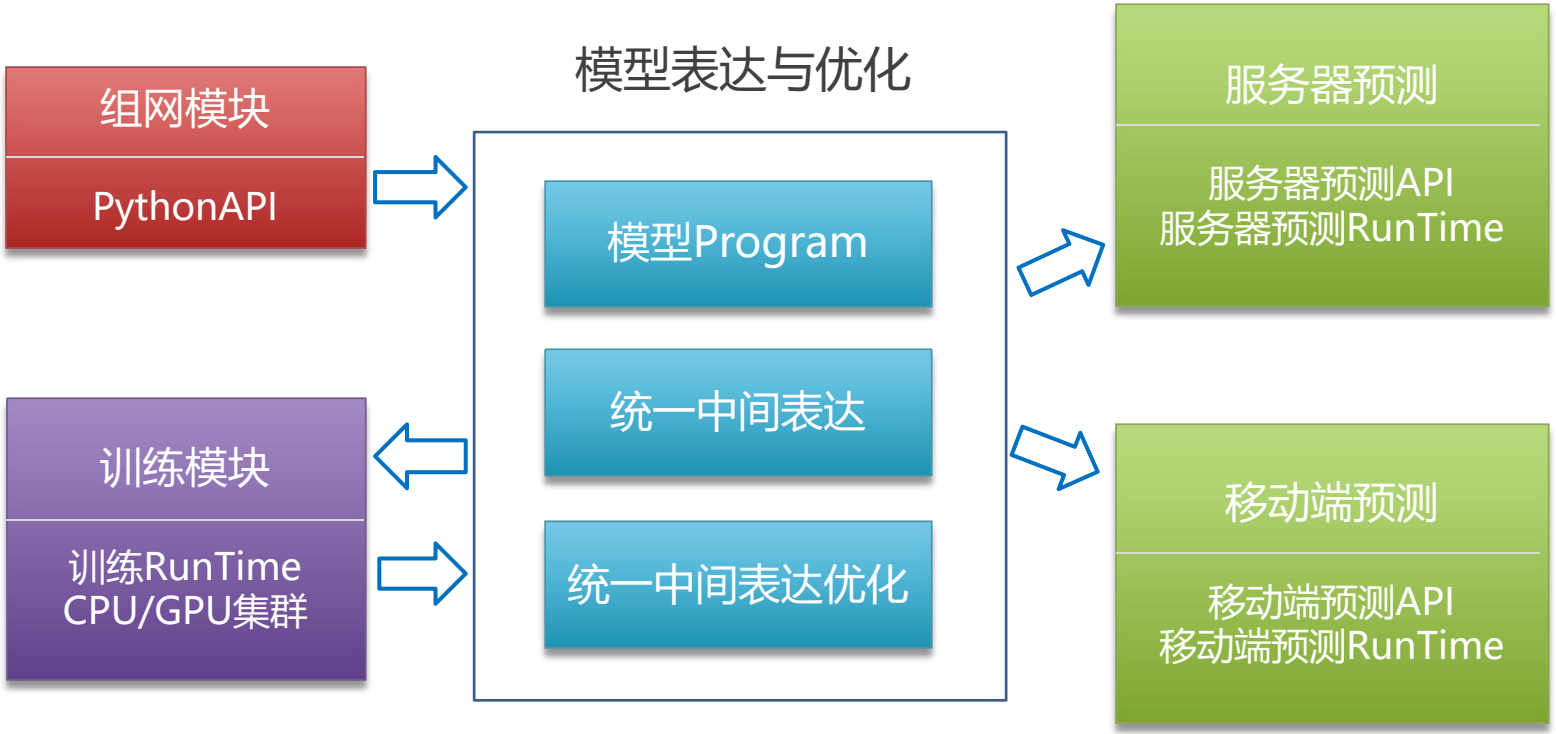


PaddlePaddle



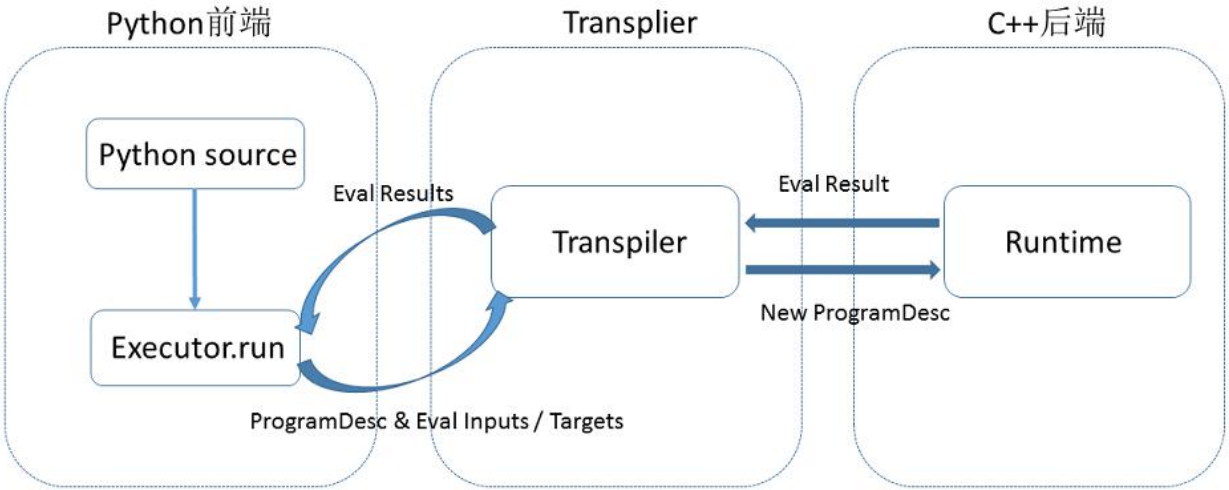
PaddlePaddle体系结构

体系架构

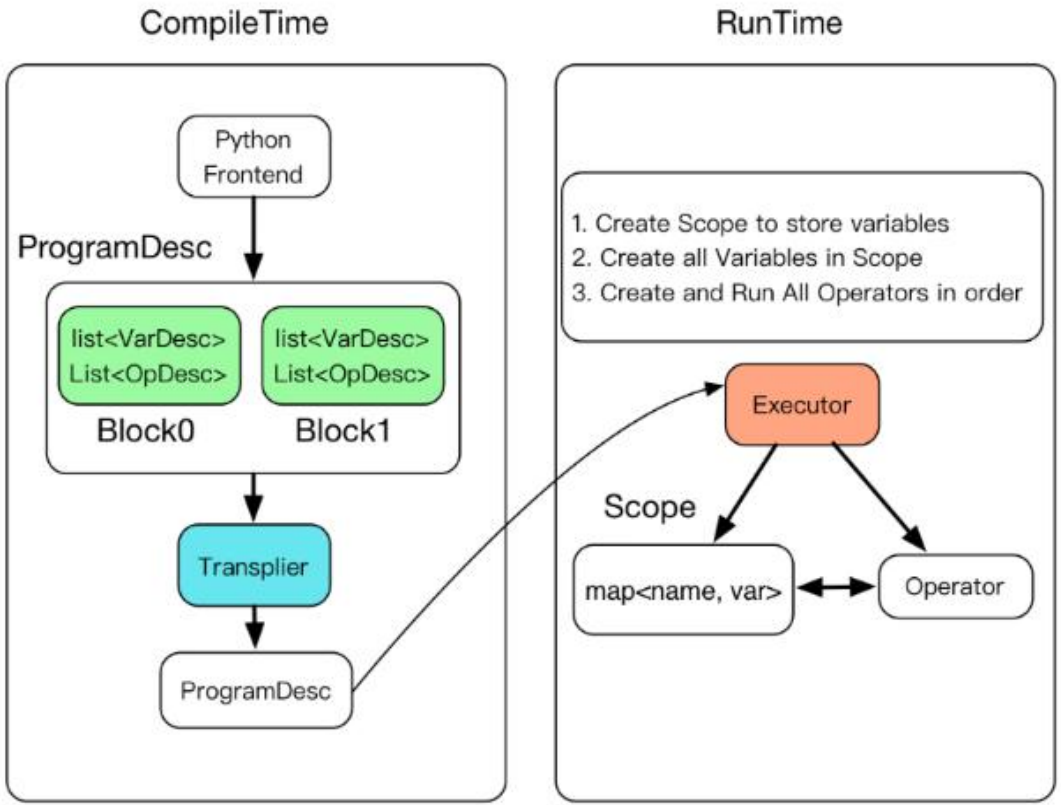

组网模块
PythonAPI
模型表达与优化
模型Program
统一中间表达
统一中间表达优化
训练模块
训练RunTime
CPU/GPU集群
服务器预测
服务器预测API
服务器预测RunTime
移动端预测
移动端预测API
移动端预测RunTime

两种语言

- Python：编写代码，设定数据及计算规则；显示结果
- 底层C++库：执行代码



编译与执行


CompileTime
RunTime
Python Frontend
ProgramDesc
list<VarDesc> List<OpDesc>
list<VarDesc> List<OpDesc>
Block0
Block1
Transplier
ProgramDesc
1. Create Scope to store variables
2. Create all Variables in Scope
3. Create and Run All Operators in order
Executor
Scope
map<name, var>
Operator

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

PaddlePaddle的”HelloWorld”

```python
import paddle.fluid as fluid

# 创建两个类型为int64，形状为1行1列的张量
x = fluid.layers.fill_constant(shape=[1], dtype="int64", value=5)
y = fluid.layers.fill_constant(shape=[1], dtype="int64", value=1)
z = x + y
# print(z) # z为对象，此时还没有值

# 创建Executor执行器
place = fluid.CPUPlace()  # 指定在CPU上运行
exe = fluid.Executor(place)  # 创建执行器

result = exe.run(fluid.default_main_program(), fetch_list=[z])
print(result[0][0])  # result为1*1的张量
```

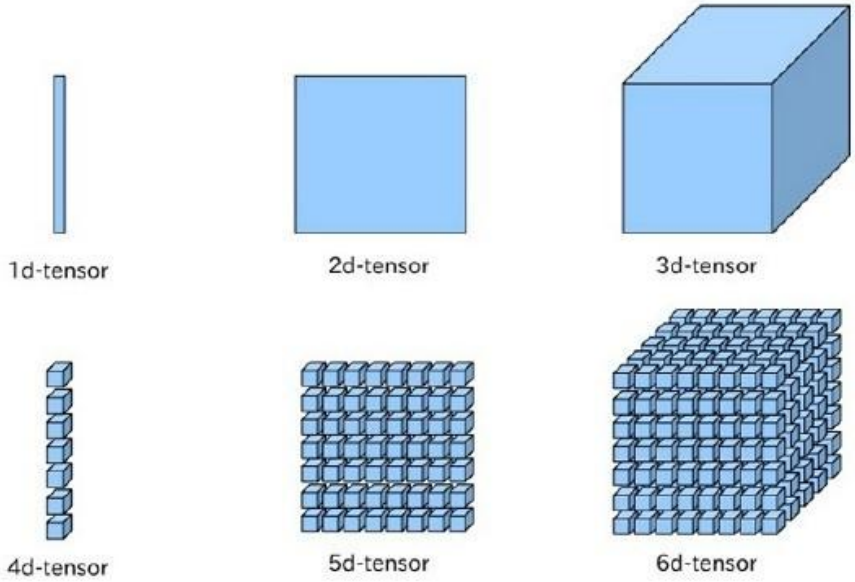
学习资源

- **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

PaddlePaddle基本概念

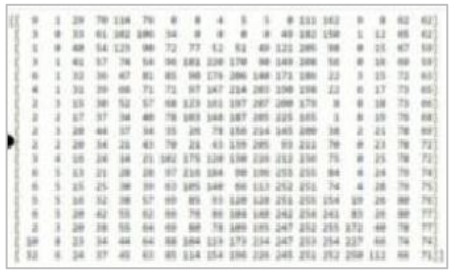
Tensor(张量)

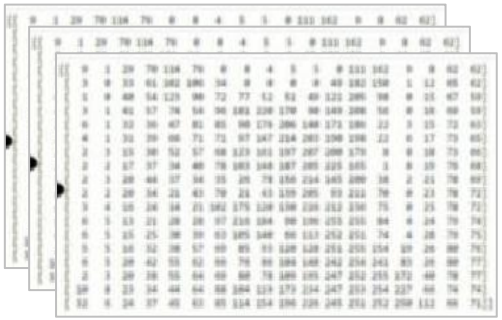
- Tensor: 多维数组或向量，同其它主流深度学习框架一样，Fluid使用张量来承载数据



Tensor示例

- 灰度图像为二维张量（矩阵），彩色图像为三维张量





思考：

一个句子是几维张量？

一篇文章是几维张量？

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（变量）

- 表示一个变量，可以是一个张量（Tensor），也可以是其它类型。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")
```

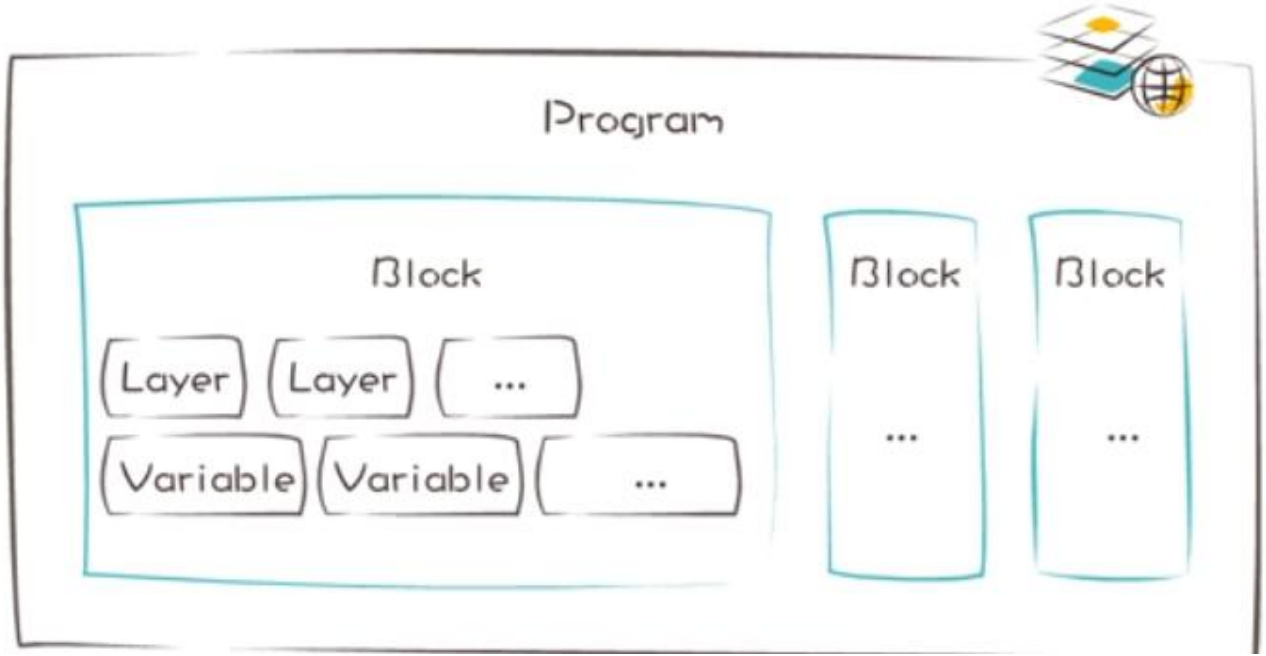
Python变量

Paddle变量

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。

代码单元关系



Executor（执行器）

- 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）。

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

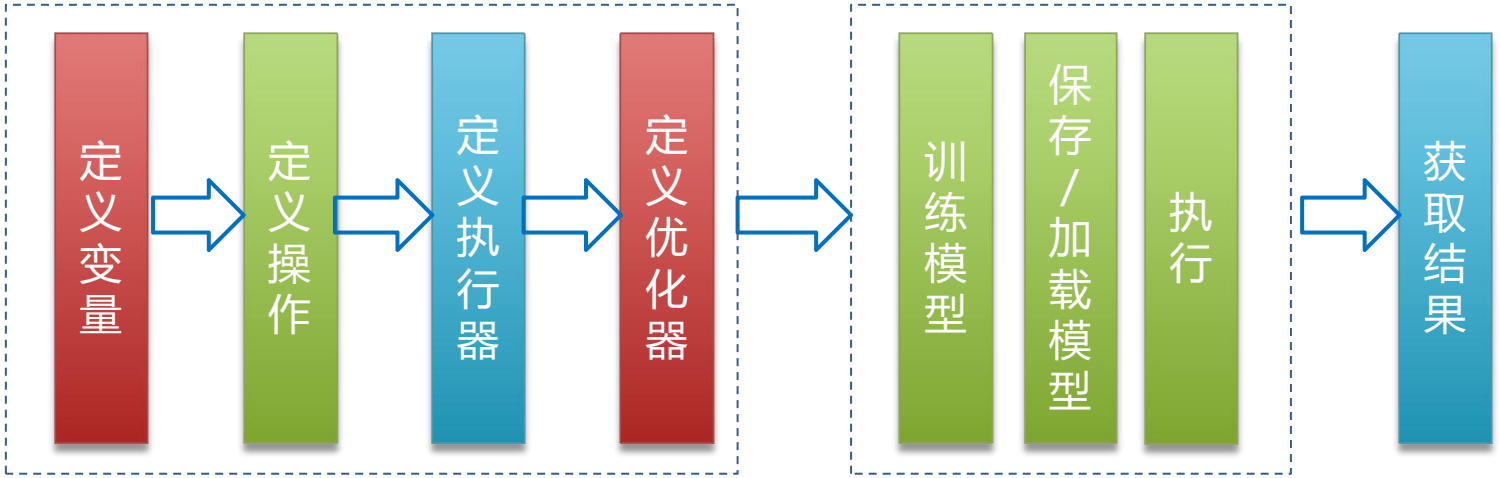
Optimizer

- 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

示例：执行两个张量计算

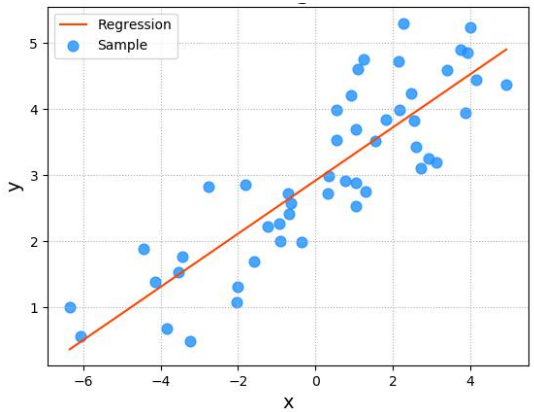
```
import paddle.fluid as fluid
import numpy
# 创建x，y两个1行1列，类型为float32的变量(张量)
x = fluid.layers.data(name="x", shape=[1], dtype="float32")
y = fluid.layers.data(name="y", shape=[1], dtype="float32")

result = fluid.layers.elementwise_add(x, y) # 两个张量按元素相加

place = fluid.CPUPlace() # 指定在CPU上执行
exe = fluid.Executor(place) # 创建执行器
exe.run(fluid.default_startup_program()) # 初始化网络

a = numpy.array([int(input("x:"))]) #输入x，并转换为数组
b = numpy.array([int(input("y:"))]) #输入y，并转换为数组

params = {"x": a, "y": b}
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
print(outs[0][0])
```

总结：程序执行步骤



示例：编写简单线性回归

- 任务：
  - ✓ 给出输入样本：**[[1.0], [2.0], [3.0], [4.0]]**
  - ✓ 给出实际输出样本：**[[2.0], [4.0], [6.0], [8.0]]**
  - ✓ 找出 $y = wx$ 公式中的 w
- 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



- 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.mean( )*作为损失函数
  - ✓ 优化器：随机梯度下降优化器 *fluild.SGD*，做梯度下降计算

代码见：simple_lr.py

关键代码（一）

- 数据准备

```
train_data = np.array([[1.0], [2.0], [3.0], [4.0]]).astype('float32')  # 输入样本
y_true = np.array([[2.0], [4.0], [6.0], [8.0]]).astype('float32')  # 实际输出样本
```

关键代码（二）

- 搭建网络

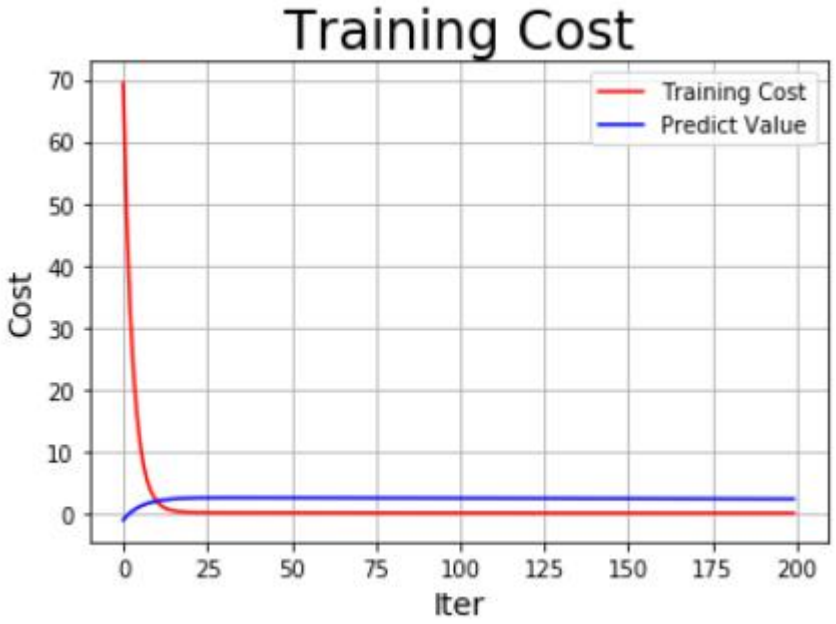
```
# 通过全连接网络进行预测
y_preict = fluid.layers.fc(input=x,   # 输入
                           size=1,   # 输出结果个数
                           act=None)   # 激活函数
# 添加损失函数
cost = fluid.layers.square_error_cost(input=y_preict, label=y)
avg_cost = fluid.layers.mean(cost)   # 求均方差
# 定义优化方法
optimizer = fluid.optimizer.SGD(learning_rate=0.01)
optimizer.minimize(avg_cost)   # 指定最小化均方差值

# 搭建网络
place = fluid.CPUPlace()   # 指定在CPU执行
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())   # 初始化系统参数
```

关键代码（三）

- 执行训练

```
# 开始训练
costs = []
iters = []
values = []
params = {"x": train_data, "y": y_true}
for i in range(200):
    outs = exe.run(feed=params, fetch_list=[y_preict.name, avg_cost.name])
    iters.append(i)  # 迭代次数
    costs.append(outs[1][0])  # 损失值
    values.append(outs[0][0])  # 预测值

    print("avg_cost:", outs[1])  # y_preict
```

训练过程


Training Cost
Training Cost
Predict Value
Cost
Iter
70
60
50
40
30
20
10
0
0
25
50
75
100
125
150
175
200

PaddlePaddle

数据准备

数据准备

什么是数据准备

为什么需要数据准备

数据准备的三种模式

如何使用数据读取器

案例：波士顿房价预测

PaddlePaddle数据准备

什么是数据准备

- 数据准备是指将样本数据从外部（主要指文件）读入，并且按照一定方式（随机、批量）传递给神经网络，进行训练或测试的过程
- 数据准备包含三个步骤：
  - ✓ 第一步：自定义Reader生成训练/预测数据
  - ✓ 第二步：在网络配置中定义数据层变量
  - ✓ 第三步：将数据送入网络进行训练/预测

为什么需要数据准备

- **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）

读取数据的三种模式

- 可以有三种模式来读取数据：reader、reader creator和reader decorator
  - ✓ reader：从本地、网络、分布式文件系统HDFS等读取数据，也可随机生成数据，并返回一个或多个数据项
  - ✓ reader creator：一个返回reader的函数
  - ✓ reader decorator：装饰器，可组合一个或多个reader

如何使用reader（示例一）

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

```
import numpy
import paddle

def reader_creator(file_path):
    def reader():
        with open(file_path, "r") as f:
            lines = f.readlines()
            for line in lines:
                yield line
    return reader

reader = reader_creator("test.txt")
for data in reader():
    print(data, end="")
```

如何使用reader（示例二）

- 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="")
```

如何使用reader（示例三）

- 从上一个随机读取器中，分批次读取数据

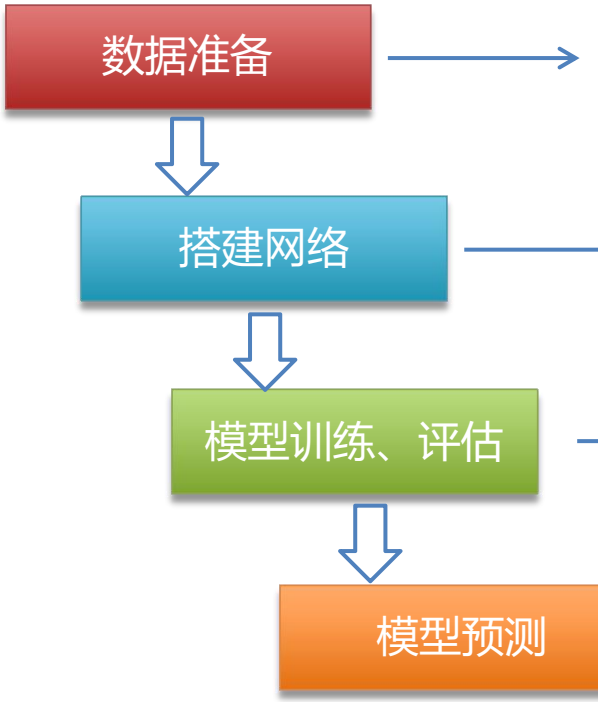
```
1   import numpy
2   import paddle
3
4   def reader_creator(file_path):
5       def reader():
6           with open(file_path, "r") as f:
7               lines = f.readlines()
8               for line in lines:
9                   yield line
10      return reader
11
12  reader = reader_creator("test.txt")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

机器学习案例：波士顿房价预测

- 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，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

思路：



使用reader对象，随机、批量读取数据样本

两个reader分别读取训练、测试样本

使用全连接网络模型，输入为13个特征值

输出为一个预测价格中位数

均方差损失函数（回归问题）

使用随机梯度下降优化器优化

训练迭代N轮

加载模型，喂入测试数据

进行测试

代码见：linear_regression.py

关键代码（一）

- 训练、测试数据准备

```
# 训练数据集读取器
random_reader = paddle.reader.shuffle(paddle.dataset.uci_housing.train(),
                                      buf_size=BUF_SIZE)  # 创建随机读取器
train_reader = paddle.batch(random_reader, batch_size=BATCH_SIZE)  # 训练数据读取器

# 测试数据集读取器
random_tester = paddle.reader.shuffle(paddle.dataset.uci_housing.test(),
                                      buf_size=BUF_SIZE)
test_reader = paddle.batch(random_tester, batch_size=BATCH_SIZE)
```

关键代码（二）

- 配置网络

```
# 定义输入、输出，类型均为张量
x = fluid.layers.data(name="x", shape=[13], dtype="float32")
y = fluid.layers.data(name="y", shape=[1], dtype="float32")
# 定义个简单的线性网络，连接输出层、输出层
y_predict = fluid.layers.fc(input=x,  # 输入数据
                            size=1,  # 输出值个数
                            act=None)  # 激活函数
# 定义损失函数，并将损失函数指定给优化器
cost = fluid.layers.square_error_cost(input=y_predict,  # 预测值，张量
                                      label=y)  # 期望值，张量
avg_cost = fluid.layers.mean(cost)  # 求损失值平均数
optimizer = fluid.optimizer.SGDOptimizer(learning_rate=0.001)  # 使用随机梯度下降优化器
opts = optimizer.minimize(avg_cost)  # 优化器最小化损失值
```

关键代码（三）

- 执行训练、保存模型

```
for pass_id in range(EPOCH_NUM):
    train_cost = 0
    i = 0
    for data in train_reader():
        i += 1
        train_cost = exe.run(program=fluid.default_main_program(),
                             feed=feeder.feed(data),
                             fetch_list=[avg_cost])
        if i % 20 == 0:  # 每40笔打印一次损失函数值
            print("PassID: %d, Cost: %0.5f" % (pass_id, train_cost[0][0]))
        iter = iter + BATCH_SIZE  # 加上每批次笔数
        iters.append(iter)  # 记录笔数
        train_costs.append(train_cost[0][0])  # 记录损失值

# 保存模型
if not os.path.exists(model_save_dir):  # 如果存储模型的目录不存在，则创建
    os.makedirs(model_save_dir)
fluid.io.save_inference_model(model_save_dir,  # 保存模型的路径
                              ["x"],  # 预测需要喂入的数据
                              [y_predict],  # 保存预测结果的变量
                              exe)  # 模型
```

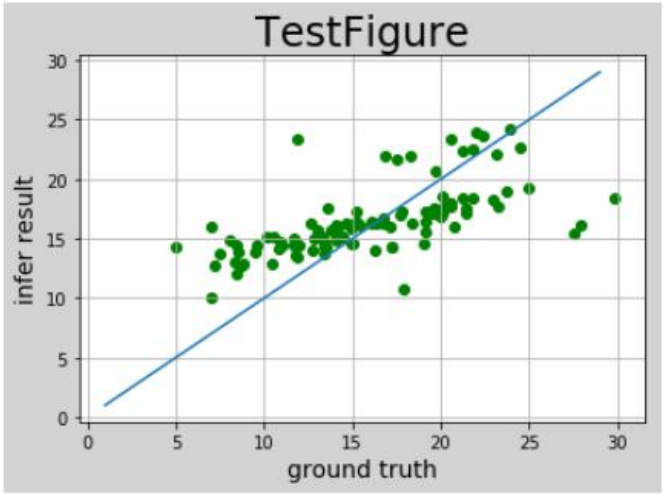
关键代码（四）

- 加载模型并预测

```
with fluid.scope_guard(infer_scope):
    [infer_program, feed_target_names, fetch_targets] = \
        fluid.io.load_inference_model(model_save_dir,  # 模型保存路径
                                      infer_exe)  # 运行模型的Executor
    # 获取测试数据
    infer_reader = paddle.batch(paddle.dataset.uci_housing.test(),
                                batch_size=200)  # 测试数据读取器
    test_data = next(infer_reader())  # 获取一条数据
    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")

    x_name = feed_target_names[0]  # 模型中保存的输入参数名称
    results = infer_exe.run(infer_program,  # 预测program
                            feed={x_name: np.array(test_x)},  # 喂入预测的值
                            fetch_list=fetch_targets)  # 预测结果
```

执行结果

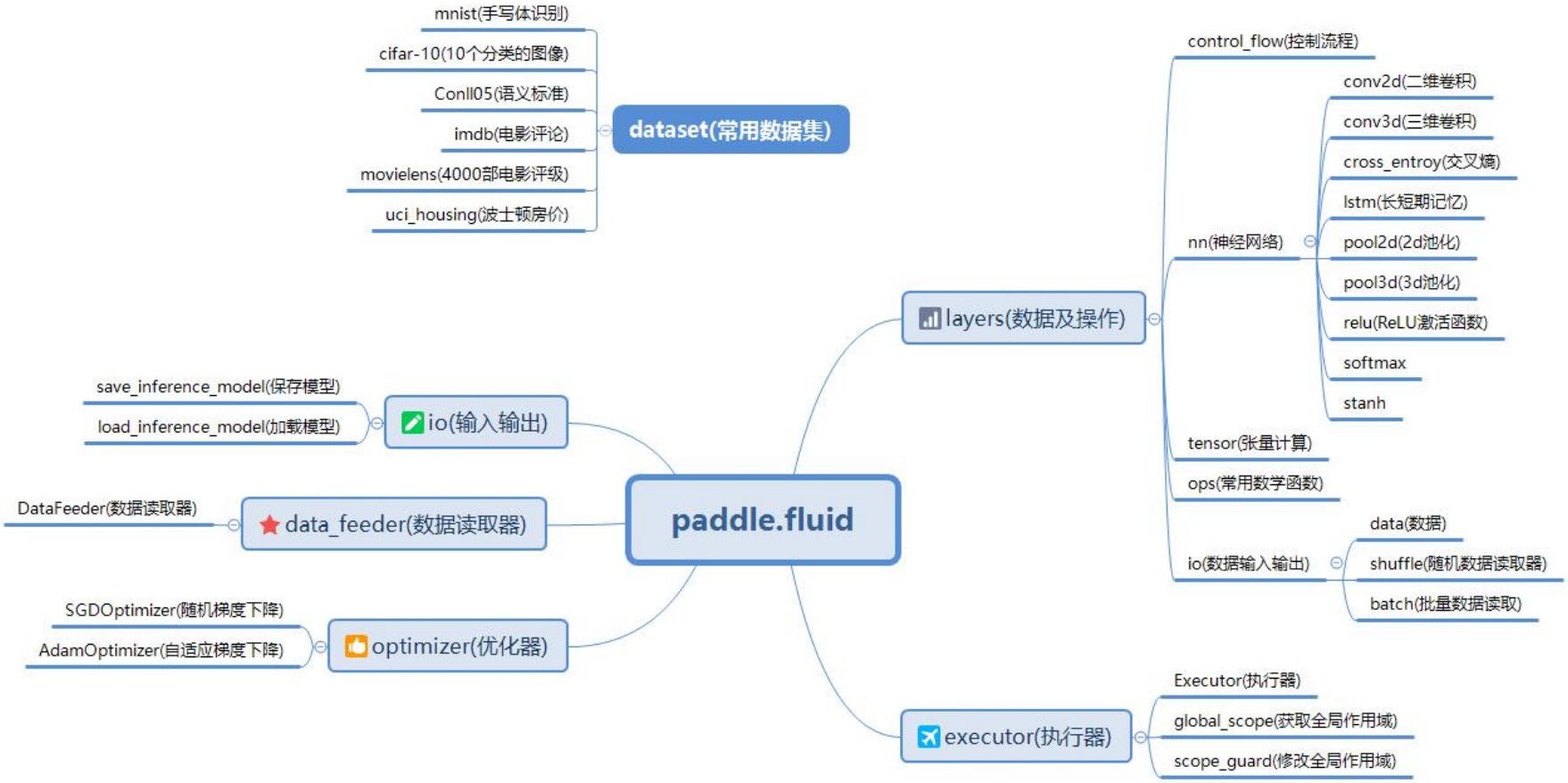
损失函数收敛过程



预测值与实际值对比

Paddle.fluid API结构

fluid API结构图



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