# 数据导入dataloader

## PyReader

说明：PyReader对象用于为计算图(Program)输入数据。根据iterable设置为True或者False，可以指定该PyReader对象是作为计算图中的一个Operator或是独立于计算图。创建PyReader对象后，采用装饰器设置其数据源。可以采用 `decorate_sample_generator(sample_generator, batch_size, drop_last=True, places=None)` , `decorate_sample_list_generator(reader, places=None)` , `decorate_batch_generator(reader, places=None)` 三种不同的装饰器。

### 初始化

```
class paddle.fluid.io.PyReader(feed_list=None, capacity=None, use_double_buffer=Tru
e, iterable=True, return_list=False)
```

参数：

- feed_list (list(Variable)|tuple(Variable)): 需要feed的变量列表，是由 `fluid.layers.data()` 创建的占位符。
- capacity (int): PyReader对象内部维护队列的容量大小。单位是batch数量。若reader读取速度较快，建议设置较大的capacity值。
- use_double_buffer (bool): 是否使用 double_buffer_reader 。设置为True时，PyReader会异步地预读取下一个batch的数据，可加速数据读取过程，但同时会占用少量的CPU/GPU存储，即一个batch输入数据的存储空间。
- iterable (bool): 所创建的DataLoader对象是否可迭代。iterable=False时，该PyReader对象作为计算图中的一个Operator，见代码示例1。 当iterable=True时，该PyReader对象独立于计算图，是一个可迭代的python生成器，见代码示例2。
- return_list (bool): 每个设备上的数据是否以list形式返回。仅在iterable = True模式下有效。若return_list = False，每个设备上的返回数据均是str -> LoDTensor的映射表，其中映射表的key是每个输入变量的名称。若return_list = True，则每个设备上的返回数据均是list(LoDTensor)。推荐在静态图模式下使用return_list = False，在动态图模式下使用return_list = True。

返回: 被创建的reader对象

返回类型： reader (Reader)

代码示例：

1. iterable=False，创建的PyReader对象作为一个Operator将被插入到计算图(program)中。在训练时，用户应该在每个epoch之前调用 `start()` ，并在epoch结束时捕获 `Executor.run()` 抛出的 `fluid.core.EOFException` 。一旦捕获到异常，用户应该调用 `reset()` 手动重置reader。

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

EPOCH_NUM = 3
ITER_NUM = 5
BATCH_SIZE = 3

def network(image, label):
    # 用户定义网络，此处以softmax回归为例
    predict = fluid.layers.fc(input=image, size=10, act='softmax')
    return fluid.layers.cross_entropy(input=predict, label=label)

def reader_creator_random_image_and_label(height, width):
    def reader():
        for i in range(ITER_NUM):
            fake_image = np.random.uniform(low=0,
                                           high=255,
                                           size=[height, width])
            fake_label = np.ones([1])
            yield fake_image, fake_label
    return reader

image = fluid.layers.data(name='image', shape=[784, 784], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')

reader = fluid.io.PyReader(feed_list=[image, label],
                           capacity=4,
                           iterable=False)

user_defined_reader = reader_creator_random_image_and_label(784, 784)
reader.decorate_sample_list_generator(
    paddle.batch(user_defined_reader, batch_size=BATCH_SIZE))

loss = network(image, label)
executor = fluid.Executor(fluid.CPUPlace())
executor.run(fluid.default_startup_program())
```

```
for i in range(EPOCH_NUM):
    reader.start()
    while True:
        try:
            executor.run(feed=None)
        except fluid.core.EOFException:
            reader.reset()
            break
```

1. iterable=True，创建的PyReader对象与计算图(Program)分离。Program中不会插入任何算子。在本例中，创建的reader是一个可迭代的python生成器。直接用for循环将每次迭代的数据feed进Executor中，`Executor.run(feed=...)`。

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

EPOCH_NUM = 3
ITER_NUM = 5
BATCH_SIZE = 10

def network(image, label):
     # 用户定义网络，此处以softmax回归为例
     predict = fluid.layers.fc(input=image, size=10, act='softmax')
     return fluid.layers.cross_entropy(input=predict, label=label)

def reader_creator_random_image(height, width):
    def reader():
        for i in range(ITER_NUM):
            fake_image = np.random.uniform(low=0, high=255, size=[height, width]),
            fake_label = np.ones([1])
            yield fake_image, fake_label
    return reader

image = fluid.layers.data(name='image', shape=[784, 784], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
reader = fluid.io.PyReader(feed_list=[image, label], capacity=4, iterable=True, ret
urn_list=False)

user_defined_reader = reader_creator_random_image(784, 784)
reader.decorate_sample_list_generator(
    paddle.batch(user_defined_reader, batch_size=BATCH_SIZE),
    fluid.core.CPUPlace())
```

```
loss = network(image, label)
executor = fluid.Executor(fluid.CPUPlace())
executor.run(fluid.default_startup_program())

for _ in range(EPOCH_NUM):
    for data in reader():
        executor.run(feed=data, fetch_list=[loss])
```

### 装饰器

```
decorate_sample_generator(sample_generator, batch_size, drop_last=True, places=None
)
```

参数:

- sample_generator (generator) — Python生成器，yield 类型为 tuple(numpy.ndarray)
- batch_size (int) — batch size，必须大于0
- drop_last (bool) — 如果设置为True，当最后一个batch中的样本数量小于batch size时，丢弃最后一个batch
- places (None|list(CUDAPlace)|list(CPUPlace)) — 位置列表。当PyReader可迭代时必须被提供

代码示例:

```
def random_image_and_label_generator(height, width):
    def generator():
        for i in range(ITER_NUM):
            fake_image = np.random.uniform(low=0,
                                           high=255,
                                           size=[height, width])
            fake_label = np.array([1])
            yield fake_image, fake_label
    return generator

image = fluid.layers.data(name='image', shape=[784, 784], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
reader = fluid.io.PyReader(feed_list=[image, label], capacity=4, iterable=True)
user_defined_generator = random_image_and_label_generator(784, 784)
reader.decorate_sample_generator(user_defined_generator,
                                 batch_size=BATCH_SIZE,
                                 places=[fluid.CPUPlace()])
```

```
decorate_sample_list_generator(reader, places=None)
```

参数：

- reader (generator) — Python生成器，yield 类型为 list[tuple(numpy.ndarray)]， list的长度为 batch size。可以采用 `paddle.batch(reader, batch_size, drop_last=False)` 将yield 类型为 tuple(numpy.ndarray)的reader转换为此处需要的reader
- places (None|list(CUDAPlace)|list(CPUPlace)) — 位置列表。当PyReader可迭代时必须被提供

代码示例：

```
def random_image_and_label_generator(height, width):
    def generator():
        for i in range(ITER_NUM):
            fake_image = np.random.uniform(low=0,
                                           high=255,
                                           size=[height, width])
            fake_label = np.ones([1])
            yield fake_image, fake_label
    return generator

image = fluid.layers.data(name='image', shape=[784, 784], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
reader = fluid.io.PyReader(feed_list=[image, label], capacity=4, iterable=True)

user_defined_generator = random_image_and_label_generator(784, 784)
reader.decorate_sample_list_generator(
    paddle.batch(user_defined_generator, batch_size=BATCH_SIZE),
    fluid.core.CPUPlace())
```

```
decorate_batch_generator(reader, places=None)
```

参数

- reader (generator) — Python生成器， yield 类型为 tuple(numpy.ndarray) 或 tuple(LoDTensor), 其中 numpy.ndarray 或LoDTensor的shape应包含batch size 这一维度。
- places (None|list(CUDAPlace)|list(CPUPlace)) — 位置列表。当PyReader可迭代时必须被提供

代码示例：

```
def random_image_and_label_generator(height, width):
    def generator():
        for i in range(ITER_NUM):
            batch_image = np.random.uniform(low=0,
                                            high=255,
                                            size=[BATCH_SIZE, height, width])
            batch_label = np.ones([BATCH_SIZE, 1])
            batch_image = batch_image.astype('float32')
            batch_label = batch_label.astype('int64')
            yield batch_image, batch_label
    return generator

image = fluid.layers.data(name='image', shape=[784, 784], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
reader = fluid.io.PyReader(feed_list=[image, label], capacity=4, iterable=True)

user_defined_generator = random_image_and_label_generator(784, 784)
reader.decorate_batch_generator(user_defined_generator, fluid.CPUPlace())
```

## DataFeeder

**API属性：声明式编程(静态图)专用API**

### class paddle.fluid.DataFeeder(feed_list, place, program=None)

DataFeeder 负责将reader(读取器)返回的数据转成一种特殊的数据结构，使它们可以输入到 Executor 和 ParallelExecutor 中。 reader通常返回一个minibatch条目列表。在列表中每一条目都是一个样本（sample）,它是由具有一至多个特征的列表或元组组成的。
以下是简单用法：

```
import paddle.fluid as fluid
place = fluid.CPUPlace()
img = fluid.layers.data(name='image', shape=[1, 28, 28])
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
feeder = fluid.DataFeeder([img, label], fluid.CPUPlace())
result = feeder.feed([([0] * 784, [9]), ([1] * 784, [1])])
```

在多GPU模型训练时，如果需要提前分别向各GPU输入数据，可以使用 decorate_reader 函数。

```
import paddle
import paddle.fluid as fluid

place=fluid.CUDAPlace(0)
data = fluid.layers.data(name='data', shape=[3, 224, 224], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')

feeder = fluid.DataFeeder(place=place, feed_list=[data, label])
reader = feeder.decorate_reader(
      paddle.batch(paddle.dataset.flowers.train(), batch_size=16), multi_devices=Fa
lse)
```

### 参数

**feed_list (list)** — 向模型输入的变量表或者变量表名
**place (Place)** — place表明是向GPU还是CPU中输入数据。如果想向GPU中输入数据, 请使用 fluid.CUDAPlace(i) (i 代表 the GPU id)；如果向CPU中输入数据, 请使用 fluid.CPUPlace()
**program (Program)** — 需要向其中输入数据的Program。如果为None, 会默认使用 default_main_program()。 缺省值为None

### 抛出异常

**ValueError** — 如果一些变量不在此 Program 中

### 代码示例

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

place = fluid.CPUPlace()

def reader():
    yield [np.random.random([4]).astype('float32'), np.random.random([3]).astype('f
loat32')],

main_program = fluid.Program()
startup_program = fluid.Program()

with fluid.program_guard(main_program, startup_program):
      data_1 = fluid.layers.data(name='data_1', shape=[1, 2, 2])
```

```
      data_2 = fluid.layers.data(name='data_2', shape=[1, 1, 3])
      out = fluid.layers.fc(input=[data_1, data_2], size=2)
      # ...

feeder = fluid.DataFeeder([data_1, data_2], place)

exe = fluid.Executor(place)
exe.run(startup_program)
for data in reader():
    outs = exe.run(program=main_program,
                   feed=feeder.feed(data),
                   fetch_list=[out])
```

##方法
####__init__(self, feed_list, place, program=None)
初始化DataFeeder类，根据feed_list内的变量名，从program.block中获取到变量var，将各变量的数据类型、数据名、lod等级、形状，存储为列表备用。
####参数
**feed_list** 变量列表，可以是reader(读取器)返回的数据
**place** 数据转化存储的位置
**program** 决定从哪个program的block获取数据

###feed(iterable)

根据feed_list（数据输入表）和iterable（可遍历的数据）提供的信息，将输入数据转成一种特殊的数据结构，使它们可以输入到 Executor 和 ParallelExecutor 中。

####参数
iterable (list|tuple) – 要输入的数据
####返回
转换结果
####返回类型
dict
####代码示例
```
import numpy.random as random
import paddle.fluid as fluid

def reader(limit=5):
    for i in range(limit):
        yield random.random([784]).astype('float32'), random.random([1]).astype('in
t64'), random.random([256]).astype('float32')
```

```
data_1 = fluid.layers.data(name='data_1', shape=[1, 28, 28])
data_2 = fluid.layers.data(name='data_2', shape=[1], dtype='int64')
data_3 = fluid.layers.data(name='data_3', shape=[16, 16], dtype='float32')
feeder = fluid.DataFeeder(['data_1','data_2', 'data_3'], fluid.CPUPlace())

result = feeder.feed(reader())
```
###feed_parallel(iterable, num_places=None)
该方法获取的多个minibatch，并把每个minibatch提前输入进各个设备中。

####参数
**iterable (list|tuple)** – 要输入的数据
**num_places (int)** – 设备数目。默认为None。
####返回
转换结果
####返回类型
dict
####注解
设备（CPU或GPU）的数目必须等于minibatch的数目

####代码示例
```
import numpy.random as random
import paddle.fluid as fluid

def reader(limit=10):
    for i in range(limit):
        yield [random.random([784]).astype('float32'), random.random([1]).astype('f
loat32')],

x = fluid.layers.data(name='x', shape=[1, 28, 28])
y = fluid.layers.data(name='y', shape=[1], dtype='float32')

fluid.layers.elementwise_add(x, y)

feeder = fluid.DataFeeder(['x','y'], fluid.CPUPlace())
place_num = 2
places = [fluid.CPUPlace() for x in range(place_num)]
data = []
exe = fluid.Executor(fluid.CPUPlace())
exe.run(fluid.default_startup_program())
program = fluid.CompiledProgram(fluid.default_main_program()).with_data_parallel(pl
aces=places)
for item in reader():
    data.append(item)
```

```
    if place_num == len(data):
        exe.run(program=program, feed=list(feeder.feed_parallel(data, place_num)),
fetch_list=[])
        data = []
```

###decorate_reader(reader, multi_devices, num_places=None, drop_last=True)
将reader返回的输入数据batch转换为多个mini-batch，之后每个mini-batch都会被输入进各个设备（CPU或GPU）中。
####参数
**reader (fun)** – 该参数是一个可以生成数据的函数
**multi_devices (bool)** – bool型，指明是否使用多个设备
**num_places (int)** – 如果 multi_devices 为 True ，可以使用此参数来设置GPU数目。如果 multi_devices 为 None ，该函数默认使用当前训练机所有GPU设备。默认为None。
**drop_last (bool)** – 如果最后一个batch的大小比 batch_size 要小，则可使用该参数来指明是否选择丢弃最后一个batch数据。 默认为 True
####返回
转换结果
####返回类型
dict
####抛出异常
**ValueError** – 如果 drop_last 值为False并且data batch与设备不匹配时，产生此异常
####代码示例

```
import numpy.random as random
import paddle
import paddle.fluid as fluid

def reader(limit=5):
    for i in range(limit):
        yield (random.random([784]).astype('float32'), random.random([1]).astype('i
nt64')),

place=fluid.CPUPlace()
data = fluid.layers.data(name='data', shape=[1, 28, 28], dtype='float32')
label = fluid.layers.data(name='label', shape=[1], dtype='int64')

feeder = fluid.DataFeeder(place=place, feed_list=[data, label])
reader = feeder.decorate_reader(reader, multi_devices=False)

exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())
for data in reader():
    exe.run(feed=data)
```

```

###\_get\_number\_of\_places\_(self, num_places)
返回设备参数数量，GPU环境下返回GPU数，否则返回CPU数，传入设备数量则将其转换为整数返回。
####参数
**num_places** 设备数量
####返回
设备数量
```