In [1]:
```python
import torch
```

尽量不要自己给自己赋值

错误例子：自己给自己赋值

In [2]:
```python
eta1 = torch.nn.Parameter(torch.rand(10), requires_grad=True)
eta1
```
Out[2]:
```
Parameter containing:
tensor([0.0312, 0.6983, 0.4285, 0.4792, 0.2871, 0.7417, 0.1106, 0.9623, 0.9452,
        0.4848], requires_grad=True)
```

这里本来想让 `eta1` 乘以2

In [3]:
```python
eta1 = eta1 * 2
```

In [4]:
```python
eta1
```
Out[4]:
```
tensor([0.0625, 1.3966, 0.8570, 0.9584, 0.5741, 1.4835, 0.2212, 1.9247, 1.8905,
        0.9695], grad_fn=<MulBackward0>)
```

In [5]:
```python
eta1.sum().backward()
```

结果可以看出没有梯度了

In [6]:
```python
eta1.grad
```
```
/Users/haibinzhao/miniconda3/envs/ML/lib/python3.8/site-packages/torch/_tensor.py:1104: UserWarning: The .grad attribute
of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute won't be populated during autograd.backward
(). If you indeed want the .grad field to be populated for a non-leaf Tensor, use .retain_grad() on the non-leaf Tensor.
If you access the non-leaf Tensor by mistake, make sure you access the leaf Tensor instead. See github.com/pytorch/pytor
ch/pull/30531 for more informations. (Triggered internally at  /Users/distiller/project/conda/conda-bld/pytorch_16467559
22314/work/build/aten/src/ATen/core/TensorBody.h:475.)
  return self._grad
```

正确例子：赋值给其他变量名

In [7]:
```python
eta2 = torch.nn.Parameter(torch.rand(10), requires_grad=True)
eta2
```
Out[7]:
```
Parameter containing:
tensor([0.8627, 0.4348, 0.9095, 0.5124, 0.3881, 0.6686, 0.7270, 0.6214, 0.1460,
        0.6846], requires_grad=True)
```

In [8]:
```python
eta_temp = eta2 * 2
```

In [9]:
```python
eta_temp
```
Out[9]:
```
tensor([1.7255, 0.8697, 1.8190, 1.0247, 0.7762, 1.3372, 1.4540, 1.2428, 0.2920,
        1.3692], grad_fn=<MulBackward0>)
```

In [10]:
```python
eta_temp.sum().backward()
```

In [11]:
```python
eta2.grad
```
Out[11]:
```
tensor([2., 2., 2., 2., 2., 2., 2., 2., 2., 2.])
```

小心直接赋值其他变量

错误例子：直接赋值给别的变量后，内存里还是同一个变量

In [12]:
```python
v1 = torch.nn.Parameter(torch.rand(10), requires_grad=True)
v1
```
Out[12]:
```
Parameter containing:
tensor([0.2928, 0.7851, 0.0654, 0.9038, 0.8469, 0.8558, 0.8001, 0.8354, 0.2006,
        0.9795], requires_grad=True)
```

把 `v1` 赋值给 `a`

In [13]:
```python
a = v1[:4]
```

修改 `a` 的值

In [15]:
```python
a.data.copy_(torch.rand(4))
a
```
Out[15]:
```
tensor([0.1814, 0.2235, 0.1048, 0.1320], grad_fn=<SliceBackward0>)
```

发现 `v1` 的值也被改了

In [16]:
```python
v1
```
Out[16]:
```
Parameter containing:
tensor([0.1814, 0.2235, 0.1048, 0.1320, 0.8469, 0.8558, 0.8001, 0.8354, 0.2006,
        0.9795], requires_grad=True)
```

正确例子：用克隆的方法

In [17]:
```python
v2 = torch.nn.Parameter(torch.rand(10), requires_grad=True)
v2
```
Out[17]:
```
Parameter containing:
tensor([0.6647, 0.6194, 0.3355, 0.2664, 0.4136, 0.1148, 0.5026, 0.8328, 0.2710,
        0.9257], requires_grad=True)
```

In [18]:
```python
b = v2[:4].clone()
```

In [19]:
```python
b.data.copy_(torch.rand(4))
b
```
Out[19]:
```
tensor([0.3222, 0.6089, 0.8604, 0.0538], grad_fn=<CloneBackward0>)
```

In [20]:
```python
v2
```
Out[20]:
```
Parameter containing:
tensor([0.6647, 0.6194, 0.3355, 0.2664, 0.4136, 0.1148, 0.5026, 0.8328, 0.2710,
        0.9257], requires_grad=True)
```

生成可学习参数时先生成，最后再包装

错误例子：先生成Parameter，再变形

In [21]:
```python
t = torch.nn.Parameter(torch.randn(12,1), requires_grad=True)
t
```
Out[21]:
```
Parameter containing:
tensor([[-0.5475],
        [ 2.0614],
        [-0.0118],
        [ 1.6362],
        [ 0.0855],
        [ 0.2819],
        [-1.6987],
        [-1.1044],
        [ 1.1377],
        [ 0.4085],
        [ 1.4628],
        [-0.2824]], requires_grad=True)
```

In [23]:
```python
T = t.view(3,4)
T
```
Out[23]:
```
tensor([[-0.5475,  2.0614, -0.0118,  1.6362],
        [ 0.0855,  0.2819, -1.6987, -1.1044],
        [ 1.1377,  0.4085,  1.4628, -0.2824]], grad_fn=<ViewBackward0>)
```

In [24]:
```python
T.sum().backward()
```

In [25]:
```python
t.grad
```
```
/Users/haibinzhao/miniconda3/envs/ML/lib/python3.8/site-packages/torch/_tensor.py:1104: UserWarning: The .grad attribute
of a Tensor that is not a leaf Tensor is being accessed. Its .grad attribute won't be populated during autograd.backward
(). If you indeed want the .grad field to be populated for a non-leaf Tensor, use .retain_grad() on the non-leaf Tensor.
If you access the non-leaf Tensor by mistake, make sure you access the leaf Tensor instead. See github.com/pytorch/pytor
ch/pull/30531 for more informations. (Triggered internally at  /Users/distiller/project/conda/conda-bld/pytorch_16467559
22314/work/build/aten/src/ATen/core/TensorBody.h:475.)
  return self._grad
```

正确例子：先全部初始化完成，再包装成Parameter

In [24]:
```python
k = torch.randn(12,1)
k
```
Out[24]:
```
tensor([[-0.2693],
        [ 0.2612],
        [-0.8253],
        [ 0.5455],
        [ 1.5656],
        [-1.4120],
        [ 1.5922],
        [-1.7210],
        [ 0.8812],
        [ 0.9397],
        [-0.9860],
        [ 1.3318]])
```

In [25]:
```python
k = k.view(3,4)
```

In [26]:
```python
kp = torch.nn.Parameter(k, requires_grad=True)
```

In [27]:
```python
kp.sum().backward()
```

In [28]:
```python
kp.grad
```
Out[28]:
```
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
```

组装变量时的错误

错误例子：用tensor把变量重新拼起来

In [26]:
```python
a = torch.nn.Parameter(torch.rand(5), requires_grad=True)
```

In [27]:
```python
a0 = a[0] * 2
a1 = torch.log(a[1]+1)
a2 = a[2] ** 2
a3 = torch.nn.functional.softplus(a[3] * 2)
a4 = a[4]
a0, a1, a2, a3, a4
```
Out[27]:
```
(tensor(0.1184, grad_fn=<MulBackward0>),
 tensor(0.3061, grad_fn=<LogBackward0>),
 tensor(0.0003, grad_fn=<PowBackward0>),
 tensor(1.3874, grad_fn=<SoftplusBackward0>),
 tensor(0.5207, grad_fn=<SelectBackward0>))
```

组合成一个新变量

In [28]:
```python
A1 = torch.tensor([a0, a1, a2, a3, a4])
A1
```
Out[28]:
```
tensor([1.1843e-01, 3.0607e-01, 2.5946e-04, 1.3874e+00, 5.2072e-01])
```

由于重新用 `tensor` 包装了变量，切断了反向传播

In [29]:
```python
A1.sum().backward()
```
```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Input In [29], in <cell line: 1>()
----> 1 A1.sum().backward()

File ~/miniconda3/envs/ML/lib/python3.8/site-packages/torch/_tensor.py:363, in Tensor.backward(self, gradient, retain_gr
aph, create_graph, inputs)
    354 if has_torch_function_unary(self):
    355     return handle_torch_function(
    356         Tensor.backward,
    357         (self,),
   (...)
    361         create_graph=create_graph,
    362         inputs=inputs)
--> 363 torch.autograd.backward(self, gradient, retain_graph, create_graph, inputs=inputs)

File ~/miniconda3/envs/ML/lib/python3.8/site-packages/torch/autograd/__init__.py:173, in backward(tensors, grad_tensors,
retain_graph, create_graph, grad_variables, inputs)
    168     retain_graph = create_graph
    170 # The reason we repeat same the comment below is that
    171 # some Python versions print out the first line of a multi-line function
    172 # calls in the traceback and some print out the last line
--> 173 Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    174     tensors, grad_tensors_, retain_graph, create_graph, inputs,
    175     allow_unreachable=True, accumulate_grad=True)

RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn
```

正确例子：建立一个变量，然后把值填进去

In [30]:
```python
a
```
Out[30]:
```
Parameter containing:
tensor([0.0592, 0.3581, 0.0161, 0.5500, 0.5207], requires_grad=True)
```

In [32]:
```python
A2 = torch.zeros([5])
```

In [33]:
```python
A2[0] = a[0] * 2
A2[1] = torch.log(a[1]+1)
A2[2] = a[2] ** 2
A2[3] = torch.nn.functional.softplus(a[3] * 2)
A2[4] = a[4]
A2
```
Out[33]:
```
tensor([1.1843e-01, 3.0607e-01, 2.5946e-04, 1.3874e+00, 5.2072e-01],
       grad_fn=<CopySlices>)
```

就可以使用了

In [34]:
```python
A2.sum().backward()
```

In [35]:
```python
a.grad
```
Out[35]:
```
tensor([2.0000, 0.7363, 0.0322, 1.5006, 1.0000])
```

正确例子2： 用cat或stack拼起来

In [62]:
```python
A3 = torch.stack((a[0]*2,
                  torch.log(a[1]+1),
                  a[2]**2,
                  torch.nn.functional.softplus(a[3]*2),
                  a[4]))
```

In [63]:
```python
a.grad.zero_()
A3.sum().backward()
```

In [64]:
```python
a.grad
```
Out[64]:
```
tensor([2.0000, 0.7363, 0.0322, 1.5006, 1.0000])
```

可学习参数有相互组合时，不能填进去，只能cat/stack

在某些情况下，神经网络的可学习参数需要经过一些变化，然后继续进行计算。例如在某些问题中，可学习参数R_1, R_2需要经过一个黑箱过程才能变成η，然后η会参与后续的运算。也就是

$$[R_1, R_2] \rightarrow Black\ Box \rightarrow \eta \rightarrow NeuralNetwork$$

这个黑箱转换过程可以通过神经网络来实现，也就是

$$[R_1, R_2] \rightarrow NN \rightarrow \eta \rightarrow NeuralNetwork$$

有的时候R_1和R_2的比值是一个重要的因素。但是这个比值会随着normalization而被削弱。在这种情况下，人们往往人为添加一个特征，也就是训练一个3输入1输出的NN来拟合黑箱，也就是输入变成R_1, R_2, k，其中$k = \frac{R_1}{R_2}$。然后整个输入会被normalize成为R_1^N, R_2^N, k^N，注意$k^N \neq \frac{R_1^N}{R_2^N}$。

然而，我们明显能看出R_1^N, R_2^N, k^N只有2个自由度，也就是可学习参数最多只能有2个，否则就会产生矛盾。那么比如我们让可学习参数为R_1^N和R_2^N。然后推算出$k^N = \mathcal{N}\left\{\frac{\mathcal{D}(R_1^N)}{\mathcal{D}(R_2^N)}\right\}$，其中$\mathcal{N}(\cdot)$和$\mathcal{D}(\cdot)$表示normalization和denormalization的操作。然后把R_1^N和R_2^N和推导出来的k^N拼起来，放到NN里面计算η。然而，这个过程会出现问题：

In [69]:
```python
torch.manual_seed(0)
class Example(torch.nn.Module):
    def __init__(self):
        super().__init__()
        # learnable R1n and R2n
        self.paramn_ = torch.nn.Parameter(torch.rand(2),requires_grad=True)
        # exemplary NN convert [R1n, R2n, kn] to eta
        self.eta_estimator = torch.nn.Sequential(torch.nn.Linear(3,1))
        self.eta_estimator.train(False)
        # exemplary max and min of [R1, R2, k] for normalization and denormalization
        self.param_max = torch.tensor([2, 4, 2])
        self.param_min = torch.tensor([1, 2, 0])

    @property
    def Param(self):
        # calculate normalized R1
        paramn = torch.zeros([3])
        paramn[0] = self.paramn_[0]    # R1n
        paramn[1] = self.paramn_[1]    # R2n
        # denormalization
        param = paramn * (self.param_max - self.param_min) + self.param_min
        # calculate k=R1/R2
        param[2] = param[0] / param[1]
        # normalization
        paramn = (param - self.param_min) / (self.param_max - self.param_min)
        return paramn

    @property
    def eta(self):
        return self.eta_estimator(self.Param)
```

可以看出上述代码以R_1^N和R_2^N为可学习参数，然后先经过denormalization求出真实值R_1和R_2，然后在利用真实值求出k，再把真实的$[R_1, R_2, k]$进行normalization，就可以用于放到NN里用于计算η了：

In [70]:
```python
test = Example()
print('可学习参数：')
print(test.paramn_)
print('经过处理后的可学习参数')
print(test.Param)
print('利用可学习参数转换成的eta')
print(test.eta)
```
```
可学习参数：
Parameter containing:
tensor([0.4963, 0.7682], requires_grad=True)
经过处理后的可学习参数
tensor([0.4963, 0.7682, 0.2115], grad_fn=<DivBackward0>)
利用可学习参数转换成的eta
tensor([-0.4544], grad_fn=<AddBackward0>)
```

然而，由于 `paramn` 里的第三个参数k是前两个的组合，反向传递时会出现问题：

In [71]:
```python
test.Param.sum().backward()
```
```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Input In [71], in <cell line: 1>()
----> 1 test.Param.sum().backward()

File ~/miniconda3/envs/ML/lib/python3.8/site-packages/torch/_tensor.py:363, in Tensor.backward(self, gradient, retain_gr
aph, create_graph, inputs)
    354 if has_torch_function_unary(self):
    355     return handle_torch_function(
    356         Tensor.backward,
    357         (self,),
   (...)
    361         create_graph=create_graph,
    362         inputs=inputs)
--> 363 torch.autograd.backward(self, gradient, retain_graph, create_graph, inputs=inputs)

File ~/miniconda3/envs/ML/lib/python3.8/site-packages/torch/autograd/__init__.py:173, in backward(tensors, grad_tensors,
retain_graph, create_graph, grad_variables, inputs)
    168     retain_graph = create_graph
    170 # The reason we repeat same the comment below is that
    171 # some Python versions print out the first line of a multi-line function
    172 # calls in the traceback and some print out the last line
--> 173 Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
    174     tensors, grad_tensors_, retain_graph, create_graph, inputs,
    175     allow_unreachable=True, accumulate_grad=True)

RuntimeError: one of the variables needed for gradient computation has been modified by an inplace operation: [torch.Flo
atTensor []], which is output 0 of AsStridedBackward0, is at version 1; expected version 0 instead. Hint: enable anomaly
detection to find the operation that failed to compute its gradient, with torch.autograd.set_detect_anomaly(True).
```

问题就在于 `param` 的第三个参数k是用 `param[2]=param[0]/param[1]` 填进去的。应该用cat（高维度）或者stack（0维）拼起来。

正确例子：

In [72]:
```python
torch.manual_seed(0)
class Example(torch.nn.Module):
    def __init__(self):
        super().__init__()
        # learnable R1n and R2n
        self.paramn_ = torch.nn.Parameter(torch.rand(2),requires_grad=True)
        # exemplary NN convert [R1n, R2n, kn] to eta
        self.eta_estimator = torch.nn.Sequential(torch.nn.Linear(3,1))
        self.eta_estimator.train(False)
        # exemplary max and min of [R1, R2, k] for normalization and denormalization
        self.param_max = torch.tensor([2, 4, 2])
        self.param_min = torch.tensor([1, 2, 0])

    @property
    def Param(self):
        # calculate normalized R1
        paramn = torch.zeros([3])
        paramn[0] = self.paramn_[0]    # R1n
        paramn[1] = self.paramn_[1]    # R2n
        # denormalization
        param = paramn * (self.param_max - self.param_min) + self.param_min
        # calculate k=R1/R2
        param_temp = torch.stack([param[0], param[1], param[0]/param[1]])
        # normalization
        paramn = (param_temp - self.param_min) / (self.param_max - self.param_min)
        return paramn

    @property
    def eta(self):
        return self.eta_estimator(self.Param)
```

In [73]:
```python
test = Example()
print('可学习参数：')
print(test.paramn_)
print('经过处理后的可学习参数')
print(test.Param)
print('利用可学习参数转换成的eta')
print(test.eta)
```
```
可学习参数：
Parameter containing:
tensor([0.4963, 0.7682], requires_grad=True)
经过处理后的可学习参数
tensor([0.4963, 0.7682, 0.2115], grad_fn=<DivBackward0>)
利用可学习参数转换成的eta
tensor([-0.4544], grad_fn=<AddBackward0>)
```

In [74]:
```python
test.Param.sum().backward()
test.paramn_.grad
```
Out[74]:
```
tensor([1.1414, 0.8804])
```

其他解决方案

由于关键原因在于 `paramn` 的第三个元素是前两个元素的组合，这会使PyTorch出现问题。我们可以采取另外的解决方案：既然只有2个自由度可以学习，可以将任意一个变量detach掉，这样也可以解决此问题，但是会牺牲掉detach掉的分支的信息，因此不推荐：

In [78]:
```python
torch.manual_seed(0)
class Example(torch.nn.Module):
    def __init__(self):
        super().__init__()
        # learnable R1n and R2n
        self.paramn_ = torch.nn.Parameter(torch.rand(2),requires_grad=True)
        # exemplary NN convert [R1n, R2n, kn] to eta
        self.eta_estimator = torch.nn.Sequential(torch.nn.Linear(3,1))
        self.eta_estimator.train(False)
        # exemplary max and min of [R1, R2, k] for normalization and denormalization
        self.param_max = torch.tensor([2, 4, 2])
        self.param_min = torch.tensor([1, 2, 0])

    @property
    def Param(self):
        # calculate normalized R1
        paramn = torch.zeros([3])
        paramn[0] = self.paramn_[0]    # R1n
        paramn[1] = self.paramn_[1]    # R2n
        # denormalization
        param = paramn * (self.param_max - self.param_min) + self.param_min
        # calculate k=R1/R2
        param[2] = (param[0] / param[1]).detach()
        # normalization
        paramn = (param - self.param_min) / (self.param_max - self.param_min)
        return paramn

    @property
    def eta(self):
        return self.eta_estimator(self.Param)
```

In [79]:
```python
test = Example()
print('可学习参数：')
print(test.paramn_)
print('经过处理后的可学习参数')
print(test.Param)
print('利用可学习参数转换成的eta')
print(test.eta)
```
```
可学习参数：
Parameter containing:
tensor([0.4963, 0.7682], requires_grad=True)
经过处理后的可学习参数
tensor([0.4963, 0.7682, 0.2115], grad_fn=<DivBackward0>)
利用可学习参数转换成的eta
tensor([-0.4544], grad_fn=<AddBackward0>)
```

In [77]:
```python
test.Param.sum().backward()
test.paramn_.grad
```
Out[77]:
```
tensor([1., 1.])
```

总结

可以看出3个小实验的forward的结果都是一样的，但是第一个实验无法反向传播，后面两个可以。但是相比于第二个，第三个会把detach掉的分支的信息给忽略，这不是最优的选择。

In [ ]:

In [ ]:

In [ ]:

In [ ]: