# Pytorch技巧总结

## 1.指定使用GPU的编号：

- 1.设置当前使用的GPU设备仅为0号设备，设备名称为/gpu:0，`os.environ['CUDA_VISIBLE_DEVICES'] = "0"`；
- 2.设置当前使用的GPU设备为0，1两个设备，名称依次为/gpu:0、/gpu:1，`os.environ['CUDA_VISIBLE_DEVICES'] = "0,1"`，根据顺序表示优先使用0号设备，然后使用1号设备。
- 3.需要注意的是，指定GPU的命令需要放在和网络模型操作的最前面。

## 2.查看模型每层输出详情：

- 1.需要安装torchsummary或者torchsummaryX(pip install torchsummary);
- 2.使用示例如下：

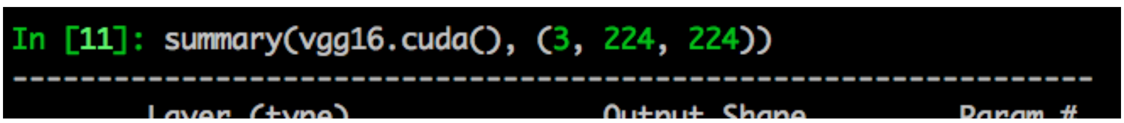
```
# 1.torchsummary使用方法
from torchsummary import summary
from torchvision import models

vgg16 = models.vgg16()
# Note: the input default as cuda, so must move the model to cuda
vgg16 = vgg16.cuda()
summary(vgg16, (3, 224, 224))    # (3, 224, 224)是网络模型的输入尺寸

# 2.torchsummaryX使用方法
from torchsummaryX import summary
from torchvision import models

vgg16 = models.vgg16()
inputx = torch.randn(1, 3, 224, 224)
summary(vgg16, inputx)
```

输出的结果如下图所示(每层输出的shape以及模型的计算量):


```
In [11]: summary(vgg16.cuda(), (3, 224, 224))
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
```

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
            Conv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
            Conv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
           Linear-32                 [-1, 4096]     102,764,544
             ReLU-33                 [-1, 4096]               0
          Dropout-34                 [-1, 4096]               0
           Linear-35                 [-1, 4096]      16,781,312
             ReLU-36                 [-1, 4096]               0
          Dropout-37                 [-1, 4096]               0
           Linear-38                 [-1, 1000]       4,097,000
================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
```

```
Forward/backward pass size (MB): 218.59
Params size (MB): 527.79
Estimated Total Size (MB): 746.96
----------------------------------------------------------------
```

## 3.梯度裁剪：防止在模型优化过程中出现梯度爆炸或弥散

```
    import torch
    import torch.nn as nn
    ...
    outputx = model(inputx)
    optimizer.zero_grad()
    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=20,
norm_type=2)
    optimizer.step()
```

nn.utils.clip_grad_norm_的参数：

- 1.parameters:基于变量的迭代器，会进行梯度归一化；
- 2.max_norm:梯度的最大范数；
- 3.norm_type:规定范数的类型，默认为L2
- 4.需要注意的是，梯度裁剪在某些任务上会额外消耗大量的计算时间。

## 4.扩展单张图片的维度：

因为在模型训练的时候，输入的数据维度是(batch_size, c, h, w)，而在测试的时候是单张图片(c，h，w)，所以需要进行维度扩展。

```python
import cv2
import torch
import numpy as np

####### 基于numpy的方法 #########
# 方法1.
image = cv2.imread(imgpath)
print(image.shape)
image = image[np.newaxis, :, :, :]
print(image.shape)

####### 基于pytorch的方法 #########
# 方法2.
image = cv2.imread(imgpath)
image = torch.tensor(image)
print(image.shape)
image = image.view(1, *image.shape)
print(image.shape)

# 方法3.
image = cv2.imread(imgpath)
image = torch.tensor(image)
print(image.shape)
image = image.unsqueeze(dim=0)
print(image.shape)
image = image.squeeze(dim=0)
print(image.shape)
```

tensor.unsqueeze(dim):扩展维度，dim指定扩展哪个维度；tensor.squeeze(dim):去除dim指定的且size为1的维度，当维度大于1时，sqeeze()不起作用，不指定dim时，去除所有size为1的维度。

## 5.one-hot编码:

在Pytorch里面的交叉熵损失函数的时候，会自动把label转换成one-hot编码，所以不需要手动转换，而使用MSE需要手动转换成one-hot编码，以下是转换示例。

```python
import torch
class_num = 8
batch_size = 4

def one_hot(label):
    """将一维列表转换为one-hot编码"""
    label = label.resize_(batch_size, 1)
    m_zeros = torch.zeros(batch_size, class_num)
    one_hot_out = m_zeros.scatter_(1, label, 1)     # (dim, index, value)
    return one_hot_out

label = torch.LongTensor(batch_size).random_() % class_num
print(one_hot(label))
```

在Pytorch1.1之后，one_hot函数可以直接调用torch.nn.functional.one_hot。

```python
import torch
import torch.nn.functional as F

tensor = torch.arange(0, 5) % 3
one_hot = F.one_hot(tensor)

# F.one_hot会检测不同类别的个数，生成对应的one-hot，也可以自己指定类别数
one_hot = F.one_hot(tensor, num_classes=10)
```

## 6.防止验证模型时显存爆炸:

在验证模型的过程中是不需要求导，即不需要梯度计算，关闭autograd，可以提高速度，节约内存，如果不关闭可能会爆显存。

```python
with torch.no_grad():
    pass
```

## 7.学习率的衰减策略:

在模型的训练过程中动态地调整学习率。

```
    import torch
    import torch.optim as optim
    from torch.optim import lr_scheduler

    # 训练前的初始化
    optimizer = optim.Adam(net.parameters(), lr=0.0001)
    scheduler = lr_scheduler.StepLR(optimizer, 10, 0.1)     # 每隔10
个epoch,学习率乘以0.1

    # 训练过程中
    for n in n_epoch:
        scheduler.step()
        ...
```

## 8.训练过程中冻结某些层的参数:

当加载预训练模型的时候，需要冻结前面几层，使其参数在训练过程中不发生变化。

```
    net = Network()
    for name, value in net.named_parameters():
    print('name: {0}, \t grad: {1}'.format(name,
value.requires_grad)

    no_grad = ['cnn.VGG_16.convolution1_1.weight',
               'cnn.VGG_16.convolution1_1.bias'
            ]
    for name, value in net.named_parameters():
        if name in no_grad:
            value.requires_grad = False
        else:
            value.requires_grad = True

    # 定义优化器
    optimizer = optim.Adam(filter(lambda p: p.requires_grad,
net.parameters()), lr=0.01)
```

## 9.训练过程中不同层设置不同的学习率:

对模型的不同层设置不同的学习率。

```
    net = Network()
    for name, value in net.named_parameters():
        print('name: {}'.format(name)

    # 根据关键词将模型的各个层分开，特征层finetune，分类层from scratch
    conv_params = []
    fc_params = []
    for name, params in net.named_parameters():
        if 'conv' in name:
            conv_params += [params]
        else:
            fc_params += [params]

    # 定义优化器
    optimizer = optim.Adam([
                {'params': conv_params, 'lr': 1e-4},
                {'params': fc_params, 'lr': 1e-2}],
weight_decay=1e-3)
```

将模型划分为两部分，存放于一个列表中，每个部分就对应上面的一个字典，在字典里设置不同的学习率。当这两部分有相同的其他参数时，就将该参数放到列表外面作为全局参数，就像上面的'weight_decay'。也可以在列表外面设置一个全局学习率，当各个部分字典里设置了局部学习率时，就使用该学习率，否则就使用列表外面的全局学习率

```
    optimizer = optim.Adam([{'params': conv_params, 'lr': 1e-4}],
lr=1e-2, weight_decay=1e-3)
```

## 10.模型的保存加载操作：

在模型的训练过程中需要对模型进行保存，使用模型的时候需要加载训练好的模型。Pytorch中保存和加载模型的主要分为两类：1. 保存加载整个模型；2. 只保存加载模型参数；

### 1.保存加载模型基本用法

#### 1.保存加载整个模型(网络结构+权重参数，比较耗时)

```
    # save model
    torch.save(model, 'net.pkl')

    # load model
    model = torch.load('net.pkl')
```

### 2.只保存加载模型参数(速度快，占内存少，推荐方法)

```
    # save model parameters
    torch.save(model.state_dict(), 'net_params.pkl'

    # load model parameters,  must build model firstly, load
parameters secondly
    model = Net()
    state_dict = torch.load('net_params.pkl')
    model.load_state_dict(state_dict)
```

**2.保存加载自定义模型**

上面保存的`net.pkl`文件其实是一个字典，通常包括以下内容： a.网络结构：输入尺寸，输出尺寸以及隐含层信息，以便能够在加载时重建模型； b.模型的权重参数：包括各个网络层训练后的可学习参数，可以在模型实例上调用`state_dict()`方法来获取，比如只保存模型权重参数时用到的`model.state_dict()`; c.优化器参数：有时候保存模型之后需要接着训练，那么就必须保存优化器的状态和所使用的超参数，也就是在优化器实例上调用`state_dict()`方法来获取这些参数； d.其他信息：有时候需要保存其他信息，比如`epoch`,`batch_size`等超参数。　　这样就可以自定义需要保存的内容，如下所示。

```python
# saving a checkpoint assuming the network class named Net
checkpoint = {
    'model':Net(),
    'model_state_dict':model.state_dict(),
    'optimizer_state_dict':optimizer.state_dict(),
    'epoch':epoch
}

torch.save(chekpoint, 'checkpoint.pkl')

# load the model infor
def load_checkpoint(filepath):
    checkpoint = torch.load(filepath)
    model = checkpoint['model']     # 网络结构
    model.load_state_dict(checkpoint['model_state_dict'])    # 加载网络模型参数
    optimizer = optim.SGD()
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])    # 加载优化器参数

    for params in model.parameters():
        params.requires_grad = False

    model.eval()
    return model

model = load_checkpoint('checkpoint.pkl')
```

加载模型是为了进行测试，则将每一层的 `requires_grad` 置为 `False`，固定这些参数；还需要调用 `model.eval()` 将模型置为测试模式，主要是将 `Dropout` 和 `BatchNormalization` 进行固定，否则模型的预测结果每次都会不同。如果继续训练，则调用 `model.train()` 确保网络模型处于训练模式。

### 3.跨设备保存加载模型；

`torch.load('net.pkl', map_location)`, map_location: a function, torch.device, string or a dict specifying how to remap storage locations.

#### 1.在GPU上训练的模型，在CPU上加载(Save on GPU, Load on CPU):

```
device = torch.device('cpu')
model = Net()
# load all tensors onto the CPU device
model.load_state_dict(torch.load('net_params.pkl',
map_location=device))
# <===> model.load_state_dict(torch.load('net_params.pkl',
map_location='cpu'))
```

### 2.在GPU上训练的模型，在GPU上加载(Save on GPU, Load on GPU):

```
device = torch.device('cuda')
model = Net()
model.load_state_dict(torch.load('net_params.pkl'))
model.to(device)
```

**在这里使用 `map_location` 参数不起作用，要使用 `model.to(torch.device("cuda"))` **将模型转换为CUDA优化的模型。

还需要对将输入模型的数据调用 `data=data.to(device)`，即将数据从CPU转到GPU。注意，调用 `my_tensor.to(device)` 会返回一个 `my_tensor` 在GPU上的副本，它不会覆盖 `my_tensor`。因此需要手动覆盖张量： `my_tensor = my_tensor.to(device)`

### 3.在CPU上训练的模型，在GPU上加载(Save on CPU, Load on GPU):

```
device = torch.device('cuda')
model = Net()
model.load_state_dict(torch.load('net_params.pkl',
map_location='cuda:0'))
model.to(device)
```

## 11.CUDA的用法，在Pytorch中和GPU相关的几个函数：

```
import torch

# 判断cuda时候可用
print(torch.cuda.is_available()

# 获取gpu数量
print(torch.cuda.device_count()

# 获取gpu名字
print(torch.cuda.get_device_name(0))

# 获取当前gpu设备索引，默认从0开始
print(torch.cuda.current_device())

# 将模型和数据从cpu移到gpu
use_cuda = torch.cuda.is_available()

# 方法1
if use_cuda:
    data = data.cuda()
    model.cuda()

# 方法2
device = torch.device('cuda' if use_cuda else 'cpu')
data = data.to(device)
model.to(device)
```

## 12.打印模型在inference中的特征图

### 方法1.包装模型(在forward中输出特征层):

```
import os
import cv2
import numpy as np
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models


class FeatureVisualizaiton:
    input_size = 256
```

```
    def __init__(self, imgpath='', layers_idx=[1, 2],
save_features_dir='/'):
        self.imgpath = imgpath
        self.layers_idx = layers_idx
        self.save_features_dir= save_features_dir
        self.net = models.vgg16()


    @staticmethod
    def preprocess_image(imgpath):
        assert os.path.isfile(imgpath), "The image of {%s} must be
existed!" % imgpath
        img = cv2.imread(imgpath)
        # resize
        img = cv2.resize(img, (input_size, input_size))
        # normalize as [0, 1]
        img = (img / 255.).astype('float32').transpose((2, 0, 1))
[np.newaxis, :, :, :]   # (1, 3, 256, 256)
        # <===>
        # img = (img / 255.).astype('float32').swapaxis(1,
2).swapaxis(0, 1)
        # img = np.expand_dims(img, axis=0)
        img = torch.from_numpy(img)
        return img


    def get_features(self):
        """Extract features"""
        features = {}
        inputx = self.preprocess_image(self.imgpath)
        print('inputx shape', inputx.shape)
        if torch.cuda.is_available():
            inputx = inputx.cuda()
            model = self.net.cuda()

        x = inputx
        for index, (name, module) in
enumerate(model.named_modules()):
            x = module(x)
            if index in self.layers_idx:
                features[name] = x
        return features


    def save_features(self):
        """Save features"""
        features = self.get_features()
        for name, feature in features.items():
```

```
            feature = self.process_feature(feature)
            cv2.imwrite(os.path.join(self.save_features_dir, name +
'.jpg'), feature)


    @statcimethod
    def process_feature(feature):
        """
        Normalize the feature
        Arguments:
            feature: (type, tensor(b, c, h, w)), normalize to (0,
255)
        """
        feature = feature.cpu().detach().numpy()

        # use sigmoid to [0, 1]
        feature = (1.0 / (1 + np.exp(-1 * feature))
        feature = np.round(feature * 255)
        return feature

if __name__ == '__main__':
    featurevisualization = FeatureVisualization()
    featurevisualization.save_features()
```

方法2.使用hook:利用pytorch里面的hook，可以不改变输入输出中间的网络结构，可以方便的获取，改变网络中间层的值和梯度(几种hook和forward，backward的先后关系在 `nn.module` 的 `__call__` 函数里面可以看得更清楚)，可以看到，对于 `register_forward_hook` 在 `forward` 的调用之后。

```python
def __call__(self, *input, **kwargs):
    for hook in self._forward_pre_hooks.values():
        result = hook(self, input)
        if result is not None:
            if not isinstance(result, tuple):
                result = (result,)
            input = result
    if torch._C._get_tracing_state():
        result = self._slow_forward(*input, **kwargs)
    else:
        result = self.forward(*input, **kwargs)
    for hook in self._forward_hooks.values():
        hook_result = hook(self, input, result)
        if hook_result is not None:
            result = hook_result
    if len(self._backward_hooks) > 0:
        var = result
        while not isinstance(var, torch.Tensor):
            if isinstance(var, dict):
                var = next((v for v in var.values() if isinstance(v, torch.Tensor)))
            else:
                var = var[0]
        grad_fn = var.grad_fn
        if grad_fn is not None:
            for hook in self._backward_hooks.values():
                wrapper = functools.partial(hook, self)
                functools.update_wrapper(wrapper, hook)
                grad_fn.register_hook(wrapper)
    return result
```

```python
import os
import cv2
import numpy as np
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models


class FeatureVisualizaiton:
     input_size = 256
     def __init__(self, imgpath='', layers_idx=[1, 2],
save_features_dir='/'):
         self.imgpath = imgpath
         self.layers_idx = layers_idx
         self.save_features_dir= save_features_dir
         self.net = models.vgg16()
```

```
    @staticmethod
    def preprocess_image(imgpath):
        assert os.path.isfile(imgpath), "The image of {%s} must be 
existed!" % imgpath
        img = cv2.imread(imgpath)
        # resize
        img = cv2.resize(img, (input_size, input_size))
        # normalize as [0, 1]
        img = (img / 255.).astype('float32').transpose((2, 0, 1))
[np.newaxis, :, :, :]   # (1, 3, 256, 256)
        # <===>
        # img = (img / 255.).astype('float32').swapaxis(1, 
2).swapaxis(0, 1)
        # img = np.expand_dims(img, axis=0)
        img = torch.from_numpy(img)
        return img


    def get_features(self):
        """Extract features"""
        features = {}
        inputx = self.preprocess_image(self.imgpath)
        print('inputx shape', inputx.shape)
        if torch.cuda.is_available():
            inputx = inputx.cuda()
            model = self.net.cuda()

        # closure
        def get_activation(name):
            def hook(model, input, output):
                features[name] = output.detach()
            return hook

        # register hook
        for layer_idx in self.layers_idx:
            handle = 
model[layer_idx].register_forward_hook(get_activation(str(layer_idx
))

        outputx = model(inputx)
        handle.remove()

        return features


    def save_features(self):
        """Save features"""
```

```
        features = self.get_features()
        for name, feature in features.items():
            feature = self.process_feature(feature)
            cv2.imwrite(os.path.join(self.save_features_dir, name +
'.jpg'), feature)


    @statcimethod
    def process_feature(feature):
        """
        Normalize the feature
        Arguments:
            feature: (type, tensor(b, c, h, w)), normalize to (0,
255)
        """
        feature = feature.cpu().detach().numpy()

        # use sigmoid to [0, 1]
        feature = (1.0 / (1 + np.exp(-1 * feature))
        feature = np.round(feature * 255)
        return feature

if __name__ == '__main__':
    featurevisualization = FeatureVisualization()
    featurevisualization.save_features()
```

## 13.Pytorch里面tensor类型之间的转换方法：主要包括下面三种方法

**1.使用独立函数；**

```
import torch
import torch.nn as nn

x = torch.randn(3, 5)
print(x)
# convert x as long
x_long = x.long()
# convert x as half
x_half = x.half()
# convert x as int
x_int = x.int()
# convert x as double
x_double = x.double()
# convert x as float
x_float = x.float()
# convert x as char
x_char = x.char()
# convert x as byte
x_byte = x.byte()
# convert x as short
x_short = x.short()
```

其中， `torch.Tensor` , `torch.rand` , `torch.randn` 均默认生成 `torch.FloatTensor` 类型。

2.使用 `torch.type()` 函数；

```
import torch
import torch.nn as nn

x = torch.randn(3, 5)
x_int = x.type(torch.IntTensor)
print(x_int)
```

3.使用 `type_as(ano_tensor)` 将tensor转换为给定类型的tensor;

```python
import torch
import torch.nn as nn

x = torch.FloatTensor(5)

y = torch.IntTensor([10, 20])

x_int = x.type_as(y)
assert isinstance(x_int, torch.IntTensor)
```