# 梯度累积

- 梯度累积：按顺序执行Mini-Batch，同时对梯度进行累积，累积的结果在最后一个Mini-Batch计算后求平均更新模型变量。

小的Batch size可能导致算法学习收敛速度慢。网络模型在每个Batch的更新将会确定下一次Batch的更新起点。每次Batch都会训练数据集中，随机抽取训练样本，因此所得到的梯度是基于部分数据噪声的估计。在单次Batch中使用的样本越少，梯度估计准确度越低。换句话说，较小的Batch size可能会使学习过程波动性更大，从本质上延长算法收敛所需要的时间。

Batch size越大，意味着神经网络训练的时候所需要的样本就越多，导致需要存储在AI芯片内存变量激增。在许多情况下，没有足够的AI加速芯片内存，Batch size设置得太大，就会出现OOM报错（Out Off Memor）。

在深度学习训练的时候，数据的batch size大小受到GPU内存限制，batch size大小会影响模型最终的准确性和训练过程的性能。在GPU内存不变的情况下，模型越来越大，那么这就意味着数据的batch size缩小，这个时候，梯度累积（Gradient Accumulation）可以作为一种简单的解决方案来解决这个问题。

梯度累积是一种训练神经网络的数据Sample样本按Batch拆分为几个小Batch的方式，然后按顺序计算。

深度学习模型由许多相互连接的神经网络单元所组成，在所有神经网络层中，样本数据会不断向前传播。在通过所有层后，网络模型会输出样本的预测值，通过损失函数然后计算每个样本的损失值（误差）。神经网络通过反向传播，去计算损失值相对于模型参数的梯度。最后这些梯度信息用于对网络模型中的参数进行更新。

梯度累积则是只计算神经网络模型，但是并不及时更新网络模型的参数，同时在计算的时候累积计算时候得到的梯度信息，最后统一使用累积的梯度来对参数进行更新。

$$accumulated = \sum_{i=0}^{N} grad_i$$

数据并行：使用多个AI加速芯片并行训练所有Mini-Batch，每份数据都在单个AI加速芯片上。累积所有Mini-Batch的梯度，结果用于在每个Epoch结束时求和更新网络参数。

```
for accu_step in range(self.__C.GRAD_ACCU_STEPS):
    optim.zero_grad()

    img_feat_iter = img_feat_iter.cuda()
    ques_ix_iter = ques_ix_iter.cuda()
```

```
    ans_iter = ans_iter.cuda()

    pred = net(
        sub_img_feat_iter,
        sub_ques_ix_iter
    )
    loss = loss_fn(pred, sub_ans_iter)
    # only mean-reduction needs be divided by grad_accu_steps
    # removing this line wouldn't change our results because the speciality of
Adam optimizer,
    # but would be necessary if you use SGD optimizer.
    # loss /= self.__C.GRAD_ACCU_STEPS
    loss.backward()
    loss_sum += loss.cpu().data.numpy() * self.__C.GRAD_ACCU_STEPS
    。。。
optim.step()
```

```
# 梯度累加参数
accumulation_steps = 4


for i, (images, labels) in enumerate(train_data):
    # 1. forwared 前向计算
    outputs = model(imgaes)
    loss = criterion(outputs, labels)

    # 2.1 loss regularization loss正则化
    loss += loss / accumulation_steps

    # 2.2 backward propagation 反向传播计算梯度
    loss.backward()

    # 3. update parameters of net
    if ((i+1) % accumulation)==0:
        # optimizer the net
        optimizer.step()
        optimizer.zero_grad() # reset grdient
```

# Linux mkdir 命令

```
# 创建目录。
# -p 确保目录名称存在，不存在的就建一个。
mkdir [-p] dirName
```

# linux wget 命令

```
# Linux wget是一个下载文件的工具
# wget默认会以最后一个符合”/”的后面的字符来命令，对于动态链接的下载通常文件名会不正确。
# -O 以指定文件名保存
wget https://github.com/explosion/spacy-
models/releases/download/en_vectors_web_lg-2.1.0/en_vectors_web_lg-2.1.0.tar.gz -O
en_vectors_web_lg-2.1.0.tar.gz
```

# Linux echo 命令

```
# echo 命令是 Linux 中最基本和最常用的命令之一。 传递给 echo 的参数被打印到标准输出中。

# echo 通常用于 shell 脚本中，用于显示消息或输出其他命令的结果。

str="Hello World"
echo "$str, good morning"
```

# YAML

YAML 是一种较为人性化的数据序列化语言，可以配合目前大多数编程语言使用。

YAML 的语法比较简洁直观，特点是使用空格来表达层次结构，其最大优势在于数据结构方面的表达，所以 YAML 更多应用于编写配置文件，其文件一般以 .yml 为后缀。

```
# model.yml
LAYER: 6
HIDDEN_SIZE: 512
MULTI_HEAD: 8
DROPOUT_R: 0.1
FLAT_MLP_SIZE: 512
FLAT_GLIMPSES: 1
FLAT_OUT_SIZE: 1024
LR_BASE: 0.0001
LR_DECAY_R: 0.2
GRAD_ACCU_STEPS: 1
CKPT_VERSION: 'small'
CKPT_EPOCH: 13
```

用缩进表示层级关系
缩进只能使用空格，不能用 TAB 字符
缩进的空格数量不重要，但是同一层级的元素左侧必须对齐

```
# YAML
one:
  two: 2
  three:
    four: 4
    five: 5

# // 以上的内容转成 JSON 后
"one": {
  "two": 2,
  "three": {
    "four": 4,
    "five": 5
  }
}
```

---

```
# 梯度置零
optim.zero_grad()
# 根据pytorch中的backward()函数的计算，当网络参量进行反馈时，梯度是被积累的而不是被替换掉；但是在每一个batch时毫无疑问并不需要将两个batch的梯度混合起来累积，因此这里就需要每个batch设置一遍zero_grad 了。
```

```
# 在pytorch中，tensor有一个requires_grad参数，如果设置为True，则反向传播时，该tensor就会自动求导。tensor的requires_grad的属性默认为False,若一个节点（叶子变量：自己创建的tensor）requires_grad被设置为True，那么所有依赖它的节点requires_grad都为True（即使其他相依赖的tensor的requires_grad = False）

# 当requires_grad设置为False时,反向传播时就不会自动求导了
# with torch.no_grad的作用：在该模块下，所有计算得出的tensor的requires_grad都自动设置为False。
a = torch.ones(2,requires_grad=True)
b = a*2
print(a, a.grad, a.requires_grad )
b.sum().backward(retain_graph = True )
print(a, a.grad, a.requires_grad )
with torch.no_grad():
    a = a + a.grad
    print(a, a.grad, a.requires_grad )
    # a.grad.zero_()
b.sum().backward(retain_graph = True )
print(a, a.grad ,a.requires_grad )
-------------------------
tensor([1., 1.], requires_grad=True) None True
tensor([1., 1.], requires_grad=True) tensor([2., 2.]) True
tensor([3., 3.]) None False
tensor([3., 3.]) None False
```