朱浩泽 1911530
智能计算系统作业一
March 7, 2022

智能计算系统第一次作业

Question 1. 假设只有一个隐层的多层感知机, 其输入、隐层、输出层的神经元个数分别为 33、512、10, 那么这个多层感知机总共有多少个参数是可以被训练的?

答: 有 $33 \times 512 + 512 \times 10 + 2 = 22018$ 个参数是可以被训练的。

Question 2.

$$W^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.15 & 0.30 \\ 0.25 & 0.20 & 0.35 \\ 0.10 & 0.25 & 0.15 \end{bmatrix}$$

假定输入数据 $x_1 = 0.02, x_2 = 0.04, x_3 = 0.01$
截距 $b_1 = 0.4, b_2 = 0.7$
期望输出 $y_1 = 0.9, y_2 = 0.5$

$$W^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} & w_{3,3}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.25 & 0.15 & 0.30 \\ 0.25 & 0.20 & 0.35 \\ 0.10 & 0.25 & 0.15 \end{bmatrix} W^{(2)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.40 & 0.25 \\ 0.35 & 0.30 \\ 0.01 & 0.35 \end{bmatrix}$$

答: 输入到隐藏层计算

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = w^{(1)^T} x + b_1 = \begin{bmatrix} 0.25 & 0.15 & 0.30 \\ 0.25 & 0.20 & 0.35 \\ 0.10 & 0.25 & 0.15 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.04 \\ 0.01 \end{bmatrix} + 0.4 = \begin{bmatrix} 0.416 \\ 0.4135 \\ 0.4215 \end{bmatrix}$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 1 + \frac{1}{1 + e^{-v}} = \begin{bmatrix} 1 + \frac{1}{1+e^{-0.416}} \\ 1 + \frac{1}{1+e^{-0.4135}} \\ 1 + \frac{1}{1+e^{-0.4215}} \end{bmatrix} = \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix}$$

隐含层到输出层计算

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = W^{(2)^T} h + b_2 = \begin{bmatrix} 0.40 & 0.35 & 0.01 \\ 0.25 & 0.30 & 0.35 \end{bmatrix} \begin{bmatrix} 0.6025 \\ 0.6019 \\ 0.6038 \end{bmatrix} + 0.7 = \begin{bmatrix} 1.1577 \\ 1.2425 \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} \hat{y_1} \\ \hat{y_2} \end{bmatrix} = \frac{1}{1 + e^{-z}} = \begin{bmatrix} \frac{1}{1+e^{-1.1577}} \\ \frac{1}{1+e^{-1.2425}} \end{bmatrix} = \begin{bmatrix} 0.7609 \\ 0.7760 \end{bmatrix}$$

误差

$$\begin{aligned} L(w) =& L_1 + L_2 = \frac{1}{2}(y_1 - \hat{y_1})^2 + \frac{1}{2}(y_2 - \hat{y_2})^2 \\ =& \frac{1}{2}(0.7609 - 0.9)^2 + \frac{1}{2}(0.7760 - 0.5)^2 \\ =& 0.0478 \end{aligned}$$

误差相对较大，进行反向传播更新
记 $w_{2,2}$ 为 w

$$L(w) = L_1 + L_2 = \frac{1}{2}(y_1 - \hat{y}_1)^2 + \frac{1}{2}(y_2 - \hat{y}_2)^2$$

$$\begin{aligned} \frac{\partial L(w)}{\partial w} &= \frac{\partial L(w)}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial t_1} \cdot \frac{\partial z_1}{\partial h_2} \cdot \frac{\partial h_2}{\partial v_2} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial L(w)}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial v_2} \cdot \frac{\partial v_2}{\partial w} \\ &= 0.6019 \times 0.2987 \times 0.04 \times (-0.3191 \times 0.1819 \times 0.35 + 0.2760 \times 0.7760 \times 0.2240 \times 0.3) \\ &= 0.00005307 \end{aligned}$$

$$w = w - \partial \times \frac{\partial L(w)}{\partial w} = 0.2 - 0.00005 \approx 0.20$$

朱浩泽 1911530
智能计算系统作业二
March 7, 2022

智能计算系统二次作业

Question 1. 以表 3.7 为例，计算置信度阈值为 0.3 时的精度（Precision）、召回率(Recall)、平均精度（AP）

答：当置信度阈值为 0.3 时，我们认为第 1、3、7、9、11、12、14、17、19、20 为 postive，但实际上只有第 3、5、7、10、14、18、20 为 true positive，所以

$$precision = \frac{k}{N} = \frac{P}{TP+FP} = \frac{4}{4+6} = 0.4$$
$$Recall = \frac{k}{M} = \frac{TP}{TP+FN} = \frac{4}{25} = 0.16$$

去掉置信度小于 0.3 的数据做下表：

编号	置信度	标签	召回率	精度	最大精度
3	0.92	1	1/25	1	1
7	0.78	1	2/25	1	1
11	0.69	0		2/3	
20	0.52	1	3/25	3/4	3/4
9	0.47	0		3/5	
17	0.45	0		3/6	
12	0.43	0		3/7	
1	0.35	0		3/8	
14	0.35	1	4/25	4/9	4/9
19	0.32	0		4/10	

$$AP = \frac{(1+1+3/4+4/9)}{25} \approx 0.128$$

Question 2. 通过比较已有的基于卷积神经网络的图像分类算法和图像目标检测算法，选取其中的某一方向谈一下对深度学习网络的认识；

答：

- **AlexNet**
  通过使用 ReLU 作为 CNN 的激活函数解决 Sigmoid 在网络较深时的梯度弥散问题
  通过在最后几个全连接层使用 Dropout 避免模型过拟合
  通过使用最大池化避免平均池化的模糊化效果，且步长比卷积核尺寸小
  通过 LRN 层使得响应大的值变得更大，并通过抑制反馈较小的值增强泛化能力
  通过随机从 265*256 的原始图像中截取 224*224 大小的区域进行数据增强以达到减轻过拟合、提升模型泛化能力的效果，并在预测时取图片的四个角和中间 5 个位置左右翻转，最后通过 10 次预测的结果求得平均值达到最终结果
- **VGG**
  反复堆叠 3*3 的小型卷积核和 2*2 的最大池化层，建立了 16-19 层的简洁结构
  拓展性强，迁移到其他图片数据上的泛化性非常好
  提供了非常好的初始化权重，经常被用来提取图像特征，可用于 domain Specific 的图像分类任务上进行再训练
  多个小卷积层堆叠在一起，达到了利用更参数获得更大视野的目的，而且拥有更多的非线性变换，提高了学习能力

在训练时先训练简单网络，后用简单网络的权重初始化复杂模型，加快收敛速度
预测和训练采用了 Multi-Scale 方法对数据进行增强
采用更深的神经网络以及更多的卷基层和非线性激活函数来提升分类准确率
相对于 AlexNet 来说，参数更多但需要的迭代的次数更少

- **Inception**
  V1 层层数相对较多，但参数却相对于 AlexNet 很少
  利用全局平均池化层取代最终的全连接层，使训练模型更快且减轻的过拟合
  提高了参数利用率和网络表达能力，同时可以对输出通道进行升维和降维，以达到利用很小的计算量便可以增加一层特征变换和非线性变换的目的
  在输出通道维度上通过聚合操作合并多个分支，增加了网络对不同尺寸的适应性
  通过一起发射的神经元连在一起这一特点达到学习过程中的刺激使神经元间的突触强度增强的目的

通过上述三种主流基于卷积神经网络的图像分类算法和图像目标检测算法，可以看出目前主流的深度学习网络都会通过多层的神经网络进行正向传导和负向调节，并需要训练多轮次。但随之而来的问题是一个深度学习网络相邻各个层之间的各个神经元之间也需要大量的连接，导致参数的数目大量增长，这便需要需要利用大量的时间和算力，且过深的网络以及神经元之间的全连接以及过多的训练轮次可能会导致模型过拟合，所以现行的深度学习模型主要解决优化的就是这些方面的问题，以卷机神经网络为例，其主要思想便是通过局部连接大大减少网络的参数，并在卷积层中通过每个滤波器与上一层局部连接，同时每个滤波器的所有局部连接都使用同样的参数，来达到参数共享以控制参数的数量；并通过池化逐渐降低数据的空间尺寸，再次减少网络中参数的数量，使得计算资源耗费变少，也能有效控制过拟合。

Question 3. 计算 AlexNet、VGG19、ResNet152 三个网络中神经元数目及可训练的参数数量，简述一下对于深度学习模型加速必要性的认识。

答:

- **AlexNet**
  Input：224× 224× 3=150528
  CNN1: 96 个 11× 11× 3 的卷积核，
  神经元数量为：55× 55× 48× 2=290400
  参数数量：11× 11× 3× 96+96=34944
  CNN2: 256 个 5× 5× 48 卷积核，
  神经元数量为：27× 27× 128× 2=186624
  参数数量：（5× 5× 48× 128+128）× 2=307456
  CNN3: 384 个 3× 3× 256 卷积核，
  神经元数量为：13× 13× 192× 2=64896
  参数数量：3× 3× 256× 384+384=885120
  CNN4: 384 个 3× 3× 192 卷积核，
  神经元数量为：13× 13× 192× 2=64896
  参数数量：（3× 3× 192× 192+192）× 2=663936
  CNN5: 256 个 3× 3× 192 卷积核，
  神经元数量为：13× 13× 128× 2=43264
  参数数量：（3× 3× 192× 128+128）× 2=442624
  FC1：
  神经元数量：4096
  参数数量：（6× 6× 128× 2）× 4096+4096=37752832
  FC2：
  神经元数量：4096
  参数数量：4096× 4096+4096=16781312
  FC2：

神经元数量：4096
参数数量：4096× 1000+1000=4097000
通过上面计算：神经元数量总数：809800　　参数数量总数：60965224

- **VGG19**
  同理，我们知道 VGG19 的结构是 19 层，同理上述计算可以算出神经元总数量为 $64+64+128+128+256+256+256+256+512+512+512+512+512+512+512+512+4096+4096+1000=14696$ 个，参数总数为 $3\times3\times3\times64+3\times3\times64\times64+3\times3\times128+3\times3\times64\times128+\cdots+7\times7\times512\times4096+4096\times4096+4096\times1000=143652544$ 个
- **ResNet152**
  其网络结构和神经数目如下表，由于数值过于巨大，不再进行累加计算。

网络层（操作）	输入	filter	stride	padding	输出
Input	224x224x3				224x224x3
Conv3-64	224x224x3	3x3x64	1	1	224x224x64
Conv3-64	224x224x64	3x3x64	1	1	224x224x64
MaxPool2	224x224x64	2x2	2	0	112x112x64
Conv3-128	112x112x64	3x3x128	1	1	112x112x128
Conv3-128	112x112x128	3x3x128	1	1	112x112x128
MaxPool2	112x112x128	2x2	2	0	56x56x128
Conv3-256	56x56x128	3x3x256	1	1	56x56x256
Conv3-256	56x56x256	3x3x256	1	1	56x56x256
Conv3-256	56x56x256	3x3x256	1	1	56x56x256
MaxPool2	56x56x256	2x2	2	0	28x28x256
Conv3-512	28x28x256	3x3x512	1	1	28x28x512
Conv3-512	28x28x512	3x3x512	1	1	28x28x512
Conv3-512	28x28x512	3x3x512	1	1	28x28x512
MaxPool2	28x28x512	2x2	2	0	14x14x512
Conv3-512	14x14x512	3x3x512	1	1	14x14x512
Conv3-512	14x14x512	3x3x512	1	1	14x14x512
Conv3-512	14x14x512	3x3x512	1	1	14x14x512
MaxPool2	14x14x512	2x2	2	0	7x7x512
FC1	7x7x512				4096
FC2	4096				4096
FC3	4096				1000

通过以上计算，我们可以轻易的看出，深度学习模型的神经元和参数数量庞大，且要经历多轮次的向前传递和负向反馈过程，每次都需要对每个参数进行更新，所以对于这种庞大的计算量，深度学习模型的加速就显得尤为重要。

智能计算系统第三次作业

朱浩泽 1911530 计算机科学与技术

问题一

> 请调研了解常用的图像数据预处理和数据增强方法。实现一个函数，从 ImageN-et2012_val 数据集中选择一张图片文件并读入数据，调整为（256, 256, 3）大小的图片，然后居中裁剪为(224, 224, 3）大小的图片；再实现一个函数，读入数据后居中裁剪为 (0.875 x width, 0.875* height,3）大小的图片，再调整为 (224, 224, 3）大小的图片

图像预处理和数据增强方法

- 在图像导入后，为了保证神经网络能够顺利地运行起来，且为了让训练过程变得更加流畅迅速，首先要确保几个问题：
  1. 全部样本的尺寸和通道数是一致的
  2. 图像最终以Tensor形式被输入网络
  3. 图像被恰当地归一化

  对于图像来说，所做的预处理主要就是适当的归一化处理和调整尺寸全部相同。
- 对于数据增强，其主要目的是减弱数据量不足所带来的影响，还可以提升模型的鲁棒性、为模型提供各种“不变性”、增加模型抗过拟合的能力,常见的数据增强手段如下：
  1. 添加略微修改的现有数据
  2. 从现有数据中重新合成新数据来增加数据量

图像预处理实战

- 首先我们定义一个函数，在数据集中随机选择一张图片

```
import tensorflow as tf
import pathlib
import random
import matplotlib.pyplot as plt


def chosen_img(path):
    data_root = pathlib.Path(path)
    img_list = []
    for item in data_root.iterdir():
        img_list.append(str(item))
    chosen = img_list[random.randint(0, len(img_list))]
    return chosen
```

- 我们将选中的图片以tensor的形式进行保存

```
def read_img(chosen):
    img_raw = tf.io.read_file(chosen)
    img_tensor = tf.image.decode_image(img_raw)
    return img_tensor
```

- 定义函数对张量进行处理，调整为（256, 256, 3）大小的图片

```
def reshape(img_tensor):
    img_reshape = tf.image.resize(img_tensor, [256, 256])
    if img_tensor[0][0][0] > 1:
        img_reshape /= 255
    return img_reshape
```

- 定义函数将tensor居中裁剪为(224, 224, 3）大小的图片

```
def cut(img_tensor):
    cut = tf.image.resize_with_crop_or_pad(img_tensor, 224, 224)
    if cut[0][0][0] > 1:
        cut /= 255
    return cut
```

- 读入数据后进行处理

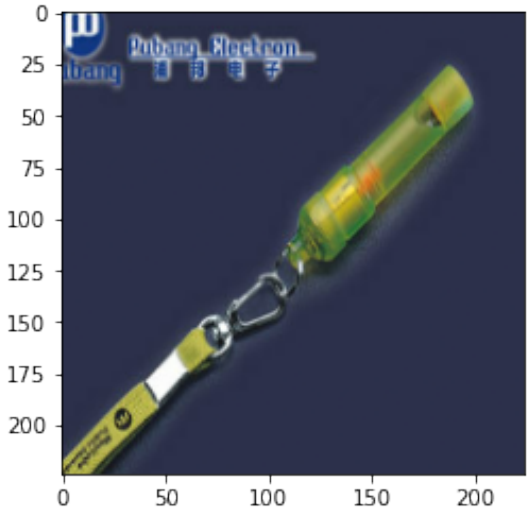
```
path = "n04579432"
img_path = chosen_img(path)
img_tensor1 = read_img(img_path)
a = reshape(img_tensor1)
a = cut(a)
img_tensor2 = read_img(img_path)
b = cut(img_tensor2)
b = reshape(b)
```

```
2022-03-17 20:24:03.731375: I tensorflow/core/platform/cpu_feature_guard.cc:145] This
TensorFlow binary is optimized with Intel(R) MKL-DNN to use the following CPU
instructions in performance critical operations:  SSE4.1 SSE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-03-17 20:24:03.732608: I tensorflow/core/common_runtime/process_util.cc:115]
Creating new thread pool with default inter op setting: 10. Tune using
inter_op_parallelism_threads for best performance.
```

- 先调整后裁剪的图片如下

```
plt.imshow(a)
```

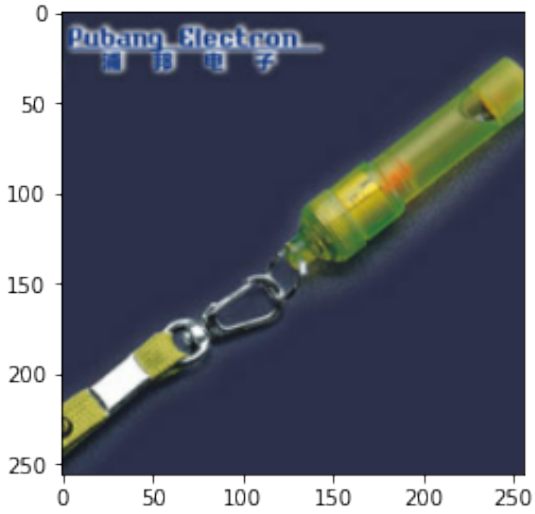
```
<matplotlib.image.AxesImage at 0x7f92aca5b410>
```



- 先裁剪后调整的图片如下

```
plt.imshow(b)
```

```
<matplotlib.image.AxesImage at 0x7f92894f8350>
```



问题二

> 查看 TensorFlow源码，在core/ops中，查找涉及conv算子的代码，请简述算子注册流程。

在Tensorflow的GitHub官方仓库 http://web.stanford.edu/class/cs20si 上寻找有关源代码，找到关于注册的文件tensorflow/tensorflow/core/ops/ragged_conversion_ops.cc和tensorflow/tensorflow/core/ops/conv_ops.cc等

首先要搞清算子的概念，数据流图中的每个节点即代表一个算子(operation)，其接收0个或多个tensor作为输入，产生0个或多个tensor作为输出。整个注册流程大约如下：

1. 用户在前端给出具体的算法以及`Input`(算子输入), `Output`(算子输出), `Attr`(算子属性)
2. 操作被转变为流图
3. 生成 `OpDefBuilderWrapper` 对象
4. 链式调用 `Input`, `Output`, `Attr` 方法并保存到 `OpDefBuilder` 的属性中
5. 对操作进行封装并作为参数传给构造函数

问题三

> 查看 TensorFlow 源码，在core/kernel 中，查找涉及conv算子的代码，请简述卷积的具体实现。

在Tensorflow的GitHub官方仓库 https://github.com/tensorflow/tensorflow 上寻找有关源代码，找到关于设计算子的代码tensorflow/tensorflow/core/kernels/conv_2d.h等

首先我们对卷积有一个初步地认识，其主要可类似看作隐藏层中的神经元 具有一个固定大小的感受视野去感受上一层的部分特征，使得在全连接神经网络中，隐藏层中的神经元的感受视野足够大乃至可以看到上一层的所有特征，其实现过成如下：

1. 通过LaunchConvOp实现具体算子的launch
2. 在具体的类中根据卷积核的大小、填充、步长等与原通道分别处理
3. 通过framework进行前置框架的引入和具体矩阵算法的加速

问题四

> 现在常见的几种机器学习框架均支持混合精度 (Mixed Precision） 训练方法，该方法采用半精度浮点做正向传播计算，使用单精度浮点做反向传播计算，在训练时需要同时存储半精度和单精度两份数据。调研了解Mixed Precision的具体实现方法，并借鉴此思想，简述如何实现稀疏卷积神经网络模型的训练。注：稀疏卷积神经网络模型一般指卷积层和全连接层的权重中有很多0元素的模型。稀疏模型采用稠密矩阵或者稀疏矩阵方法存储均可。

Mixed Precision:

1. 把weight矩阵转为FP16
2. 利用FP16矩阵（包括激活、W、激活的梯度、W的梯度）进行向前、向后计算
3. 将W的梯度全部从FP16转户为FP32，再更新总的FP32的W（因为总的W也使用FP16，则可能会由于数值指数范围较小或W的梯度的指数小于总的W的指数，在“对齐”过程中梯度很容易对齐到0）
4. 对于某些网络，需要把Loss扩大S倍（目的是把其激活的梯度扩大S倍，从而不至于其一部分数值太小变成0）并在最后W的梯度往总梯度上更新之前，把W的梯度再缩小S倍。

实现稀疏卷积网络的训练:

1. 稀疏动量通过查看最近梯度(动量)的加权平均值来确定在稀疏网络中何处增加新的权值，从而找到一致降低误差的权值和层数
2. 根据平均动量幅值大小确定各层的重要性
3. 对稀疏部分采用 float16 转化，选择性的进行权重备份
4. 对于每一层，去掉50%的最小权值，进行降维处理
5. 根据层的重要性在层之间重新分配权重。在一层中，我们在动量幅值大小较大的地方增加权重

智能计算系统第四次作业

朱浩泽 1911530 计算机科学与技术

6.3 假设设计一个深度学习处理器，它通过PCIe和DDR3的内存相连接。假设带宽为12.8Gb/s，那么和只有一个ALU的深度学习处理器相比，最理想情况下全连接层的加速比能有多少？

该深度学习处理器每个全连接层有12.8Gb的权重，12.8Gb的权重进行全连接层运算的时候一次性从DDR3中读入，然后通过并行方式运算

ALU只能一次读入一组数据进行运算，而一个ALU读入的数据量为一个字节是8b

所以加速比为 $\frac{12.8Gb}{8b} = 1.6G$

6.4 简述为什么指令级并行在深度学习处理器中一般情况下作用不大。什么情况下指令集并行也能在深度学习处理器中发挥作用？

深度学习主要是对tensor进行操作，这一点从tensorflow的名字中也可以看出。而tensor来说，其形状一般来说比较规整，而且在主要的卷积层和全连接层多为向量操作。而通过体系结构的课程我们可以知道，指令集并行性的主要优势在于灵活性高，但是流水线的控制逻辑复杂且功耗巨大，对于深度学习这种本身就功耗巨大且数据较为规整的任务不是很合适，所以只有在深度学习算法加速设计到复杂的控制逻辑时，指令集并行性才能够发挥作用，例如深度强化学习就非常需要指令集并行，用于加快速度，通常这些任务需要在某个环境中疯狂采集数据，于是就通过并行来加快采集速度；在pytorch中dataloader有一个参数叫做num_worker，就是用于开多少进程用于数据集加载，这样的好处就是可以用满机器的IO速度。这些都是指令集并行性在深度学习处理器中发挥的作用。

7.7 假设有一个单核的神经网络处理器，包含用于存放权重的片上存储WRAM共256KB，用于存放输入/输出神经元数据的片上存储NRAM共128KB，一个矩阵运算单元每个时钟周期内可完成256个32位浮点乘累加运算，该芯片运行频率为1Ghz，片外访存总带宽为64GB/s。假设运算器利用率为100%且不考虑延迟，访存带宽利用率为100%且不考虑延迟。

可以使用以下几种简化指令：

- move ram_type1 ram_type2 size，用于从 ram_type1向 ram_type2 传输size个字节的数据。其中，ram_type可选DRAM、NRAM和WRAM、
- compute compute_type num，用执行运算总量为num的compute_type类型的运算，其中，compute_type可选MAC_32、MAC_16、ADD_32、ADD_16、SUB_32、SUB_16、MUL_32、MUL_16、DIV_32、DIV_16等。
- loop loop_time ... endloop，用于表示执行循环体loop_time次。
- sync，同步指令，表示在此之前的指令必须都执行完成才能继续执行后续的指令。

请使用上述指令完成以下任务，并估计执行时间：一个全连接层，其输入的神经元个数为1X256、权重矩阵的大小为256X1，所有数据均为32位宽的浮点数。

Apache

```
compute MUL_32 256
    move WRAM DRAM 4
    move NRAM DRAM 4
    x[i,1] mul W[1,i]
sync
```

$$\text{时钟周期} = \frac{1}{1Ghz} = 1 \times 10^{-9}$$

$$\text{访存时间} = \frac{256B \times 4 \times 32 + 1B}{64GB} = 5.12 \times 10^{-7} s$$

$$\text{矩阵运算} = \frac{1}{1Ghz} = 1 \times 10^{-9}$$

$$\text{总时间} = \text{矩阵运算} + \text{访存时间} = 5.12 \times 10^{-7}$$

利用习题7.7所述的处理器和指令完成以下任务，并估算运行时间：一个全连接层，其输入神经元的个数为32x256，权重矩阵的大小为256x128，所有数据均为32位宽的浮点数。

Apache

```
loop 32
    compute MUL_32 256
    loop 128
        move WRAM DRAM 4
        move NRAM DRAM 4
        x[i,k] mul W[k,j]
    sync
    endloop
ednloop
```

$$\text{时钟周期} = \frac{1}{1Ghz}$$

$$\text{访存时间} = \frac{32 \times 32b \times 256}{64GB} = 5.12 \times 10^{-7}$$

$$\text{矩阵运算} = \frac{128 \times 32}{1Ghz} = 4.096 \times 10^{-6}$$

$$\text{执行时间} = \text{访存时间} + \text{执行时间} = 4.6 \times 10^{-6}$$