智能计算系统 课后习题

1 第1章 概述

1.1 简述强人工智能和弱人工智能的区别

人工智能大致分为两大类：强人工智能和弱人工智能。弱人工智能(weak artificial intelligence)是能够完成某种特定具体任务的人工智能，换个角度看，就是一种计算机科学非常平凡的应用。强人工智能(strong artificial intelligence)或通用人工智能，是具备与人类同等智慧，或超越人类的人工智能，能表现正常人类所具有的所有智能行为。 *(摘录自课本 Page 1)*

1.2 简述人工智能研究的三个学派

人工智能按研究学派主要分为三类，包括行为主义(Behaviorism)，符号主义(Symbolism)，连接主义(Connectionism)。 *(摘录自课本 Page 2)*

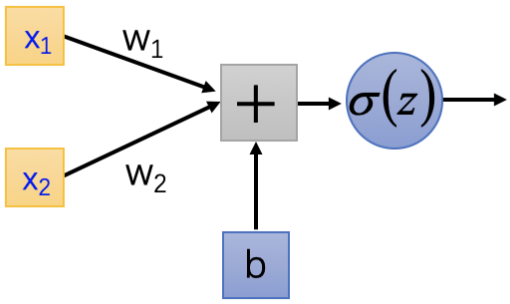
行为主义的核心思想是基于控制论构建感知-动作型控制系统。 *(摘录自课本 Page 2)* 在C.Shannon和J.McCarthy征集出版的《自动机研究》中有很多控制论方面的研究工作，涉及有有限自动机，图灵机、合成自动机，希望基于控制论去构建一些感知动作的反应性的控制系统。从比较直观的角度来看，行为主义的方法可以模拟出类似于小脑这样的人工智能，通过反馈来实现机器人的行走、抓取、平衡，因此有很大的实用价值。但是，这类方法似乎并不是通向人工智能的终极道路。 *(摘录自课本 Page 5)*

符号主义是基于符号逻辑的方法，用逻辑表示知识和求解问题。其基本思想是：用一种逻辑把各种知识都表示出来；当求解一个问题时，就将该问题转化成一个逻辑表达式，然后用已有知识的逻辑表达式的库进行推理来解决该问题。但从逻辑的角度，难以找到一种简洁的符号逻辑体系，能表述出世间所有的知识。 *(摘录自课本 Page 5)* 从常识的角度，研究者还没能把一个实用领域中的所有常识都用逻辑表达式记录下来。从求解器的角度来看，解决问题的关键环节是逻辑求解器，而各种谓词逻辑一般都是不可判定的，也就是理论上不存在一种机械方法，能在有限时间内判定任意一个谓词逻辑表达式是否成立。 *(摘录自课本 Page 6)*

连接主义方法的基本出发点是借鉴大脑中神经元细胞连接的计算模型，用人工神经网络来拟合智能行为。 *(摘录自课本 Page 6)* 连接主义方法始于1943年，从最开始的M-P神经元模型，到感知器模型、反向传播训练方法、卷积神经网络、深度学习、深度学习和反向传播训练方法，连接主义逐渐成为整个人工智能领域的主流方向。但是我们必须清楚的认识到，深度学习不一定是通向强人工智能的终极道路。它更像是一个能帮助我们快速爬到二楼、三楼的梯子，但顺着梯子我们很难爬到月球上。深度学习已知的局限性包括：泛化能力有限、缺乏推理能力、缺乏可解释性、鲁棒性欠佳等。 *(摘录自课本 Page 7 Page 8)*

1.3 一个由两个输入的单个神经元构成的感知机能完成什么任务?

首先，感知器是只有输入和输出层的神经网络:



在1958年，由美国心理学家Frank Rosenblatt提出的感知器模型中，激活函数采用的一般是符号函数，及输出

$o = sgn(x_1w_1 + x_2 * w_2 + b)$

进一步表示为：

$$\begin{cases} 1, & x_1w_1 + x_2 * w_2 + b >= 0 \\ -1, & x_1w_1 + x_2 * w_2 + b < 0 \end{cases}$$

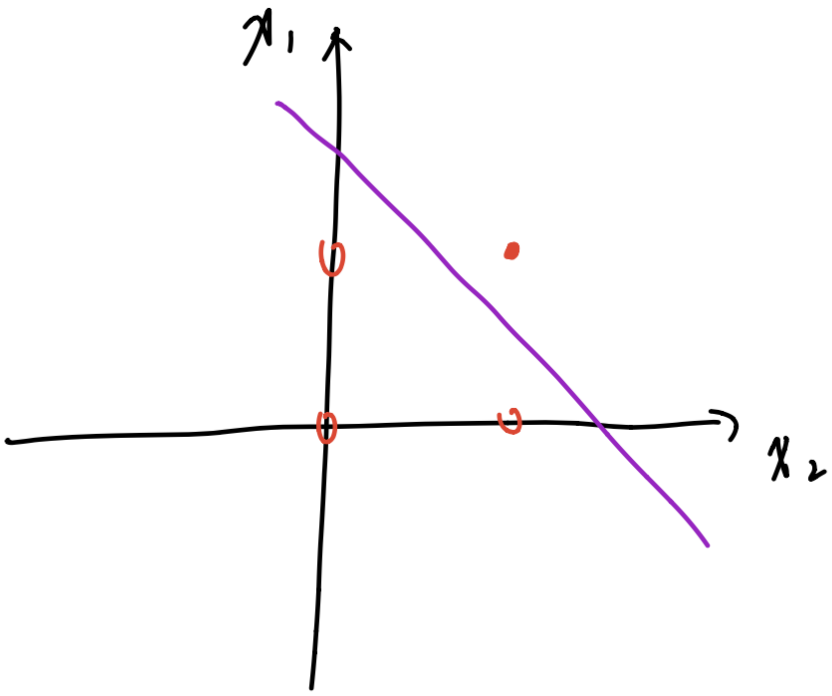
如果把o当作因变量、x_1、x_2当作自变量，对于分界

$x_1w_1 + x_2w_2 + b = 0$

可以抽象成三维空间里的一个分割面，能够对该面上下方的点进行分类，则**及该感知机能完成的任务是用简单的线性分类任务**，比如可以完成逻辑“与”与逻辑“或”的分类(在这里，第三维度只有1和-1两个值，分别使用实心点和空心点来表征，这样就可以在二维平面上将问题可视化)：

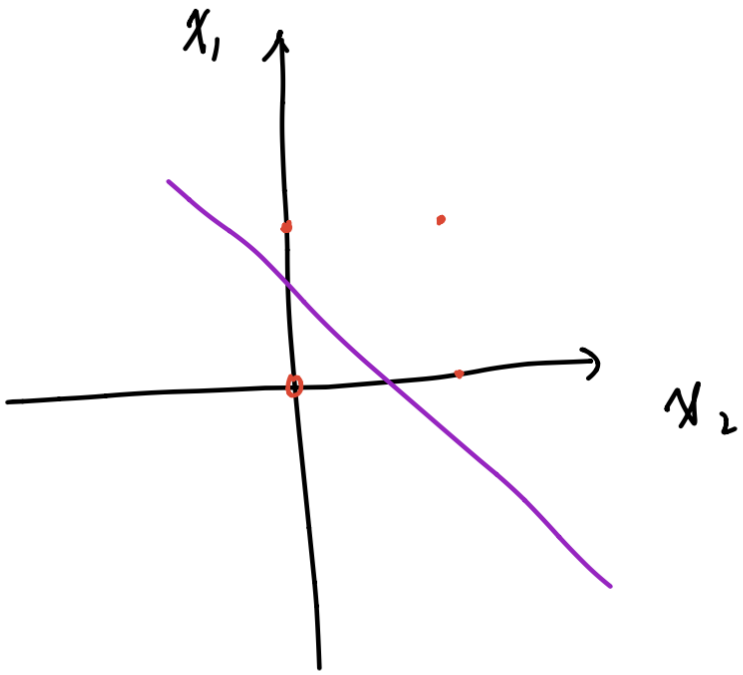
1. 逻辑“与”的真值表和二维样本：

x_1	x_2	x_3
0	0	0
0	1	0
1	0	0
1	1	1



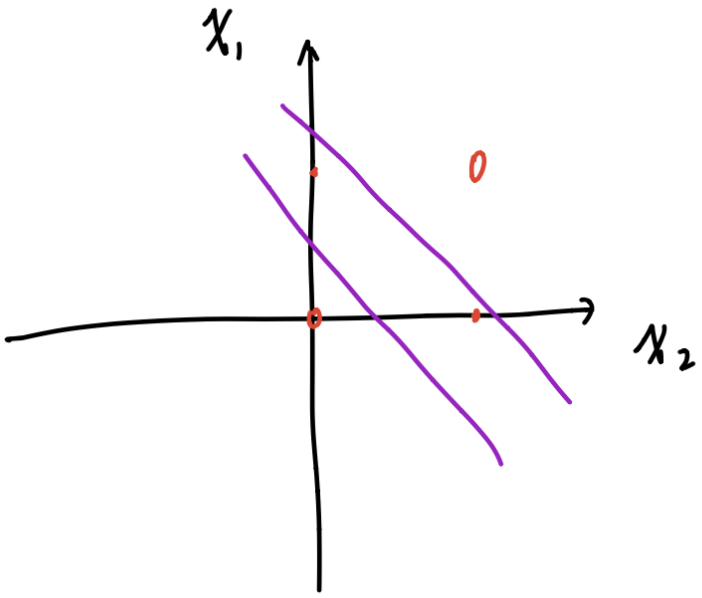
2. 逻辑“或”的真值表和二维样本:

x_1	x_2	x_3
0	0	0
0	1	1
1	0	1
1	1	1



但是对于非线性问题，如异或问题，单层感知机就没办法实现了：

x_1	x_2	x_3
0	0	0
0	1	1
1	0	1
1	1	0



多层感知机能够实现这一点，这篇博客指出了这样一个观点:**Kolmogorov理论指出： 双隐层感知器就足以解决任何复杂的分类问题.**~~但我没有找到相关的证明，就和8bit就足够表示Feature Map一样迷惑~~，在这篇文献：

- [1].APPROXIMATION CAPABILITIES OF MULTILAYER FEEDFORWARD REGULAR FUZZY NEURAL NETWORKS[J].Applied Mathematics:A Journal of Chinese Universities,2001(01):45-57.

K.Hornik证明了理论上只有一个隐层的浅层神经网络足以拟合出任意函数。但有个假设前提：一个隐层的深度可以是无限，这样就好理解了。

1.4 深度学习的局限性有哪些?

深度学习已知的局限性包括：

⑴泛化能力有限。 深度学习训练需要依靠大量的样本，与人类学习的机理不同。人类在幼儿时期会依据大量外在数据学习，但是成年人类的迁移学习能力和泛化能力远高于现在的深度学习。

⑵缺乏推理能力。 缺乏推理能力使得深度学习不擅长解决认知类的问题。如何将擅长推理的符号逻辑与深度学习结合起来，是未来非常有潜力的发展方向。

⑶缺乏可解释性。 在比较重视安全的领域，缺乏可解释性会带来一些问题。比如，某个决策是如何做出来的？深度学习为什么识别错了。

⑷鲁棒性欠佳。 在一张图像上加一些人眼很难注意到的点，就可以让深度学习算法产生错误判断，例如把猪识别成猫，把牛识别成狗。

（摘录自课本 Page 7 Page 8）

1.5 什么是智能计算系统?

智能计算系统是智能的物质载体。*（摘录自课本 Page 8）*

1.6 为什么需要智能计算系统?

传统计算系统的能效难以满足应用需求。因此，人工智能不可能依赖于传统计算系统，必须有自己的核心物质载体--智能计算系统。*（摘录自课本 Page 8）*

1.7 第一代智能计算系统有什么特点?

第一代智能计算系统主要是在20世纪80年代人工智能发展的第二次热潮中发展起来的面向符号逻辑处理的计算系统。它们的功能主要是运行当时非常热门的智能编程语言Prolog或LISP编写的程序。*（摘录自课本 Page 9）*

1.8 第二代智能计算系统有什么特点?

第二代智能计算系统主要研究面向连接主义（深度学习）处理的计算机或处理器。*（摘录自课本 Page 10）*

1.9 第三代智能计算系统有什么特点?

编者团队认为，第三代智能计算系统将不再单纯追求智能算法的加速，它将通过近乎无限的计算能力，给人类带来前所未有的机器智能。*（摘录自课本 Page 11）*

1.10 假如请你设计一个智能计算系统，你打算如何设计？在你的设计里，用户将如何使用该智能计算系统?

假如让我设计一个第二代的智能计算系统，我的方案如下：

硬件层面，借助VTA的设计思想，设计Load、Compute、Store三个模块，Load负责从DDR中缓存数据，Store负责将运算结果放回到DDR，在Compute模块里，设计一些大运算需要的加速器，例如使用脉动阵列(Systolic Array)加速矩阵乘法、快速卷机(Fast Convolution)做卷积运算等等。

软件层面:

1. 需要设计一套AI指令集，告诉加速器缓存/放回的数据地址，以及对相应数据做的Compute操作。
2. 软件上层对所需的运算，例如接受一个通用的神经网络模型，做硬件无关的优化，比如神经网络压缩，图优化，算子融合等。
3. 此外，在上层还需要做出合适的任务调度，把计算不复杂，控制性较强的部分交给CPU处理，把适合加速器运算的交给加速器运算，组成一个异构系统。
4. 软件还需要提供用户接口，可以是Python/C++等主流编程语言。

用户使用该智能计算系统时，在Python中编写相关的程序，调用配套编程框架做上层软件优化，以及对任务的调度安排，生成可以在我设计的智能计算系统上的可执行文件，用户执行该可执行文件，并给可执行文件一个输入，产生经过系统加速后的输出。

但是该系统的缺点也很明显，从底层硬件到上层软件栈，战线拉的太长了，而且硬件层面流片的成本巨大，但是深度学习所需的算子繁多并且在茁壮成长，硬件应当也需要不断的迭代更新。所以可以考虑把硬件加速的这部分换成FPGA来实现，如果需要新增算子或者更改，重新综合生成相关电路即可。

2 第2章 神经网络基础

2.1 多层感知机和感知机的区别是什么，为什么会有这样的区别?

感知机是只有一个神经元的单层神经网络。*（摘录自课本 Page 17）*

多层感知机是20世纪八九十年代常用的一种两层的神经网络。*（摘录自课本 Page 19）*

首先，感知机和多层感知机都可以接受若干个输入，但感知机只能有一个输出，多层感知机可以有多个输出。

此外，相比感知机而言，多层感知机可以解决输入非线性可分的问题。

2.2 假设有一个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中总共有多少个参数是可以被训练的?

确定weight的个数:

$$weights = 33 * 512 + 512 * 10 = 22016$$

确定bias的个数:

$$biases = 1 + 1 = 2$$

这样，总共可以被训练的参数是22018个。

2.3 反向传播中，神经元的梯度是如何计算的？权重是如何更新的?

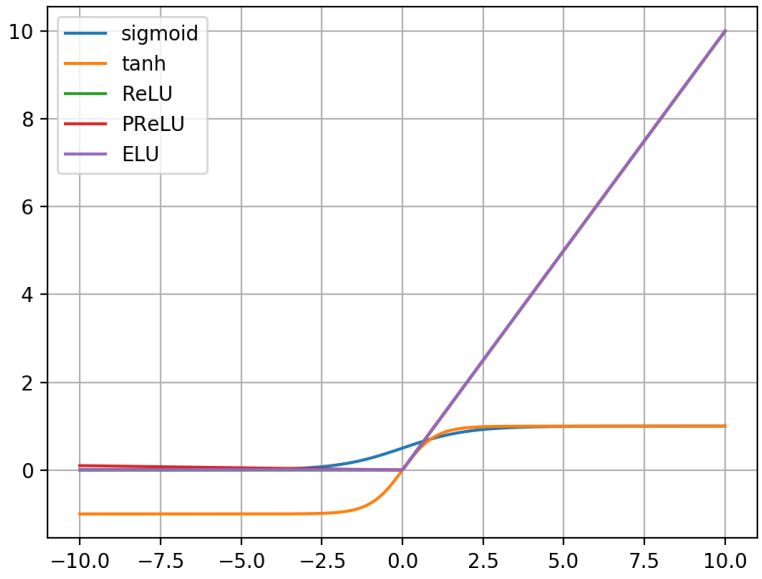
基于梯度下降法的神经网络反向传播过程首先需要根据应用场景定义合适损失函数（loss function），常用的损失函数有均方差损失函数和交叉熵损失函数。确定了损失函数之后，把网络的实际输出与期盼输出结合损失函数计算loss，如果loss不符合预期，则对loss分别求权重和偏置的偏导数，然后沿梯度下降方向更新权重及偏置参数。

不引入惩罚项的权重更新公式如下:

$$w \leftarrow w - \eta(\nabla_w L(w; x, y))$$

2.4 请在同一个坐标系内画出五种不同的激活函数图像，并比较它们的取值范围。

绘图结果如下:



对输入取区间[-10, 10],a取-0.01:比较取值范围:

Function	Minimum	Maximum
sigmoid	0	1
tanh	-1	1
ReLU	0	10
PReLU	-0.1	10
ELU	0.00038	10

2.5 请简述三种避免过拟合问题的方法

1. 在损失函数中增加惩罚项，常用的方法有L1正则化和L2正则化，L1正则化可以使训练出来的weight更接近于0，L2正则化可以使全中国weight的绝对值变小。
2. 稀疏化，在训练的时候将神经网络中的很多权重或神经元设置成0，也是通过增加一些惩罚项来实现的。
3. Bagging集成学习，应对一个问题时训练几个不同的网络，最后取结果的加权，减少神经网络的识别误差
4. Dropout会在训练的时候随机删除一些节点，往往会起到意想不到的结果，一般来说我们设置输入节点的采样率为0.8，隐层节点的采样率为0.5。在推理阶段，我们再将对应的节点输出乘以采样率，即训练的时候使用Dropout，但是推理的时候会使用未经裁剪的整个网络。

2.6 sigmoid激活函数的极限是0和1，请给出它的导数形式并求出其在原点的值

sigmoid函数如下:

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

求sigmoid函数的导数:

$$\frac{\mathrm{d}sigmoid(x)}{x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{e^{-x}+e^{x}+2}$$

在原点处的值为:

$$\frac{1}{1+1+2}=0.25$$

2.7 假设激活函数的表达式为$\phi(v)=\frac{v}{\sqrt{1+v^2}}$，请给出它的导数形式并求出其在原点的取值

$$\frac{\mathrm{d}\phi(v)}{\mathrm{d}v}=\frac{\mathrm{d}v*(1+v^2)^{-1/2}}{\mathrm{d}v}=\frac{\mathrm{d}(1+v^2)(1+v^2)^{-3/2}+\mathrm{d}-v^2(1+v^2)^{-3/2}}{\mathrm{d}v}=\frac{1}{(1+v^2)^{3/2}}$$

当在原点处时，该激活函数的导数取值为1。

2.8 假设基本采用表2.1中的符号，一个经过训练的有两个隐层的MLP如何决定各个输出神经元的标签？预测过程中，当前输入的样本的标签如何决定?

题目没有给出，两个隐藏层的神经元个数是多少个，我们可以假设输入矩阵为$\mathbf{x}$，输入层和第一层隐层的权重为$\mathbf{w1}$，第一层隐层和第二层隐层的权重为$\mathbf{w2}$.输出为$\mathbf{y}$，则有:

$$temp=w_1^T*x$$

$$y=w_2^T*temp$$

输出样本的标签由输出$\mathbf{y}$的最大值的下标决定。

2.9 一种更新权重的方法是引入动量项，公式如下，动量a的取值范围通常为[0, 1]，这样对于权重更新有什么影响？如果取值范围是[-1,0]呢?

$$\triangle w(n)=a\triangle w(n-1)+a^2\triangle w(n-2)+\ldots$$

当a为[0,1]时，动量项表现为惩罚项，每一次求梯度的时候都会考虑到之前几步更新权重的的动量，并且距离越近影响越大，梯度更新的曲线会越平滑，并且因为考虑到动量，就有可能突破local minimal，找到global minimal。具体的知识可以以`momentum`为keyword搜索相关资料。

当a为[-1,0]时，动量项表现为奖励项，这样梯度的收敛曲线应该会更陡峭，至于有啥用途想不到。

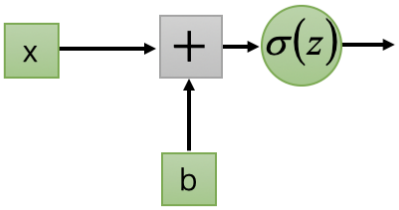
2.10 反向传播中，采用不同的激活函数对于梯度的计算有什么不同？请设计一个新的激活函数并给出神经元的梯度计算公式。

梯度的计算需要求目标函数关于权重和偏置的偏导数，激活函数不同会导致偏导数不同，进而影响梯度的计算。

设计一个新的激活函数:

$$\delta(z)=e^z$$

对一个简单的神经元:



则有

$$y = \delta(w * x + b) = e^{w*x+b}$$

选择目标函数为均方误差（mean-square error, MSE）：

$$Loss = (\check{y} - y)^2 = (\check{y} - e^{w*x+b})^2$$

神经元的梯度计算公式:

$$\nabla_w = \frac{\partial Loss}{\partial w} = -2xe^{wx+b}(\check{y} - e^{wx+b})$$

$$\nabla_b = \frac{\partial Loss}{\partial b} = 2e^{wx+b}(\check{y} - e^{wx+b})$$

2.11 请设计一个多层感知机实现4位全加器的功能，即两个4比特输入得到一个4比特输出以及1比特进位。请自行构建训练集、测试集，完成训练及测试。

Github Code Link：https://github.com/LeiWang1999/AICS-Course/tree/master/Code/2.11.fulladder.pytorch

框架：Pytorch

网络结构：简单的MLP、两个隐层layer、各20个node、激活函数使用的是Relu。

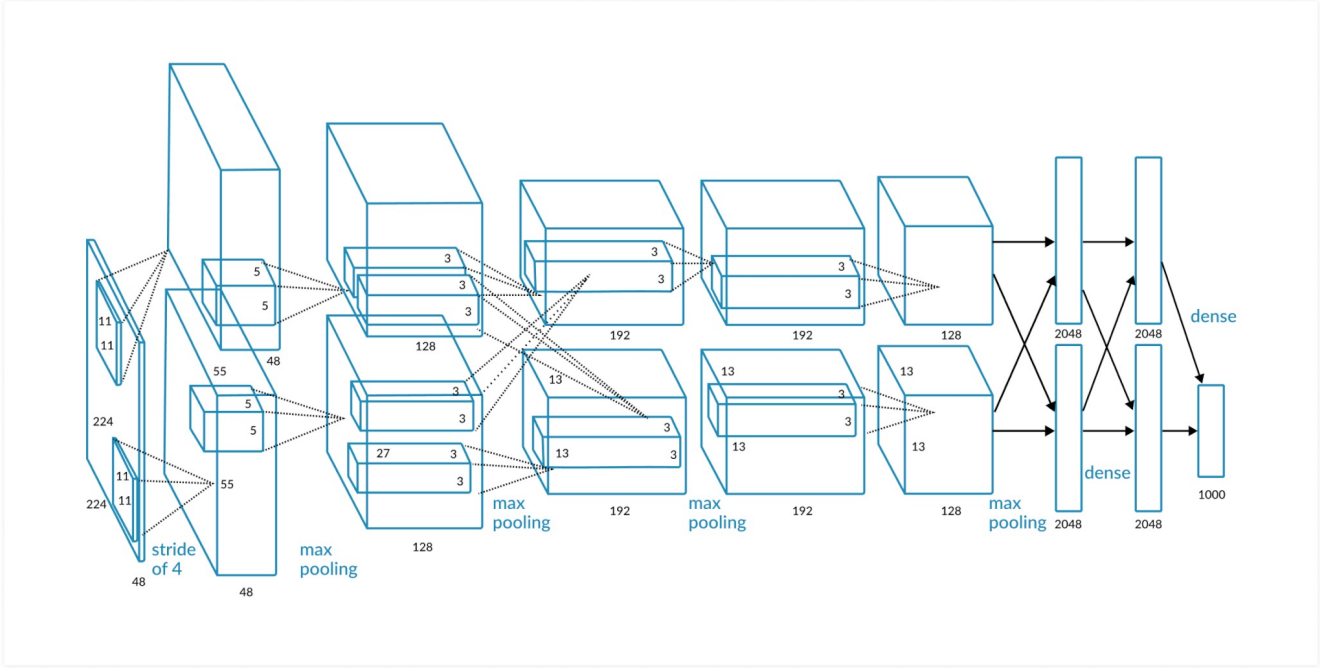
```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_1, hidden_2, output_dim):
        super(MLP,self).__init__()
        self.layer1 = nn.Sequential(nn.Linear(input_dim, hidden_1), nn.ReLU(True))
        self.layer2 = nn.Sequential(nn.Linear(hidden_1, hidden_2), nn.ReLU(True))
        self.layer3 = nn.Sequential(nn.Linear(hidden_2, output_dim))

    def forward(self,x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        return x
```

3 第3章 深度学习

3.1 计算AlexNet、VGG19、ResNet152三个网络中的神经元数目及可训练的参数数目。

3.1.1 3.1.1 AlexNet



AlexNet的论文里为了方便使用两块GPU一起训练，网络分了两路，合并成一路网络来看，共有五层卷积层，三层全联接层，但是在分析神经元个数及可训练参数的时候需要注意，使用两块GPU进行训练的网络，有部分卷积和前一级的两个特征图相连，而有部分卷积只和前一级的单块GPU上的特征图相连，具体的链接关系可以参考上图。

这里首先需要明确一下神经元的定义，题目中要计算的神经元的个数，**那对于卷积运算神经元的数目应该如何计算?**

一个神经元表示的是w*x+b的过程，而对于卷积运算来说，`feature map`取代了全连接层的layer、所以`feature map`的大小就是神经元的个数。但是对于特征图经过池化层之后得到的下一次卷积的输入，及池化后的结果不把其计算到神经元数目之中，其运算的本质是比大小，不明显符合神经元的特征。

Convolution Layer	Kernel Size	Channel Number	Output Size
conv_1	11*11	96	(27,27,96)
conv_2	5*5	256	(13,13,256)
conv_3	3*3	384	(13,13,384)
conv_4	3*3	384	(13,13,384)
conv_5	3*3	256	(6,6,256)

因为部分卷积层之后涉及到池化，所以不单独在表格后面列出神经元的数目的计算程式。

第一层是输入层、神经元的数目：

$$227 * 227 * 3 = 154587$$

无可训练参数。

对第一层卷积，因为Output Size的大小实际上是经过Max Pooling过的，所以神经元数目：

$$55 * 55 * 96 = 290400$$

可训练参数：

$$weights + bias = 96 * 11 * 11 * 3 + 96 = 34944$$

对第二层卷积，依然有Max Pooling神经元数目：

$$27 * 27 * 256 = 186624$$

可训练参数：

$$weights + bias = 2 * (128 * 48 * 5 * 5 + 128) = 307456$$

对于第三层卷积，没有Max Pooling神经元数目：

$$13 * 13 * 384 = 64896$$

可训练参数，这里是与前面的两个特征图都连接的：

$$weights + bias = 384 * 3 * 3 * 256 + 384 = 885120$$

对于第四层卷积，神经元数目：

$$13 * 13 * 384 = 64896$$

可训练参数:

$$weights + bias = 2 * (192 * 3 * 3 * 192 + 192) = 663936$$

对于第五层卷积，神经元数目：

$$13 * 13 * 256 = 43264$$

可训练参数:

$$weights + bias = 2 * (128 * 192 * 3 * 3 + 128) = 442624$$

Fully-Connect Layer	Nodes	神经元数目	可训练参数
FC1	4096	4096	$4096 * 6 * 6 * 256 + 4096 = 37752832$
FC2	4096	4096	$4096 * 4096 + 4096 = 16781312$
FC3	1000	1000	$4096 * 1000 + 1000 = 4087000$

总神经元数目：

$$154587 + 290400 + 186624 + 64896 + 64896 + 43264 + 4096 + 4096 + 1000 = 813859$$

可训练参数

$$34944 + 307456 + 885120 + 663936 + 442624 + 37752832 + 16781312 + 4087000 = 60955224$$

3.2 计算习题3.1中三个网络完成一次正向过程所需要的乘法数量和加法数量。

在3.1中，使用torchinfo计算出了所有的乘加法数量，即macs操作数。

不明白题目的意思是不是需要把乘法数量和加法数量分开来计算，如果自己计算的话，碰到指数函数怎么算乘法和加法的数量呢？

3.3 简述错误率与IoU、mAP的关系

这部分不止可以参考书上

测试结果	实际情况	
	正例(Positive)	反例(Negative)
正例(Positive)	真正例(True Positive, TP)	假正例(False Positive, FP)
反例(Negative)	假反例(False Negative, FN)	真反例(True Negative, TN)

3.4 简述训练过程中收敛、训练精度和测试精度之间的关系。

训练过程中的收敛，指的是损失函数的收敛。

每经过一次正向传播都会得到一段输出，将它与目标输出计算之后得出Loss，并且可以在训练集上得到训练精度，验证集上得到验证精度。测试精度是我们最终在未知的图片上的到的精度。

模型如果收敛了，那么训练精度就会提高，但验证精度不一定会提高。我们可以通过观察验证精度，保存在验证精度最好的模型，也可以使用early stopping技术来提前终止掉训练过程。

一般来讲，训练精度>验证精度>测试精度

3.5 试给出SVM和AlexNet在解决ImageNet图像分类问题的过程中对计算量的需求，并简述原因。

一般而言对于**计算量**，我们对于深度神经网络关注他的加法和乘法(MACs)数量，因为加法和乘法是其主要的运算，而对于SVM这类传统的机器学习方法，我们一般关注的是算法复杂度。

如果仅看推理过程，则对于小样本的分类问题，SVM和AlexNet的算法复杂度都可以看做是O(n)，但是对于复杂样本，SVM的算法复杂度会接近O(n*n)，而AlexNet的复杂度还保持在O(n)。

3.6 简述R-CNN、Fast R-CNN和Faster R-CNN的区别。

R-CNN的算法流程可以分为4个步骤：

- 一张图像生成1K～2K个候选区域（使用 Selective Search方法）
- 对每个候选区域，使用深度网络提取特征
- 特征送入每一类SVM分类器，判断是否属于该类
- 使用回归器精细修正候选框位置

Fast R-CNN算法流程可以分为3个步骤：

- 一张图像生成1K～2K个候选区域（使用 Selective Search方法）

- 将图像输入网络得到相应的特征图，将SS算法生成的候选框投影到特征图上获得相应的特征矩阵
- 将每个特征矩阵通过ROI pooling层缩放阿斗7x7大小的特征图，接着将特征图展平成一系列全联接层得到预测结果

Faster R-CNN算法流程可以分为3个步骤：

- 将图像输入网络得到相应的特征图
- 使用RPN结构生成候选框，将RPN生成的候选框投影到特征图上获得相应的特征矩阵
- 将每个特征矩阵通过ROI pooling层缩放阿斗7x7大小的特征图，接着将特征图展平成一系列全联接层得到预测结果

3.7 简述LSTM标准模型中三个门的主要作用，并给出计算公式。

1. 遗忘门单元用来控制需要记住前一时刻单元状态的多少内容，并通过sigmoid函数将遗忘门的值限制在0到1之间。第t时刻的遗忘门单元为：

$$f^{(t)} = \delta(U^f x^{(t)} + W^f h^{(t-1)} + b^f)$$

2. 输入门单元用来控制写入多少输入信息到当前状态。其计算方法与遗忘门蕾丝，也是通过sigmoid函数将取值范围限制在0到1之间。第t时刻的输入门单元为：

$$i^{(t)} = f^{(t)} = \delta(U^i x^{(t)} + W^i h^{(t-1)} + b^i)$$

3. 输出门单元可以控制当前单元状态的输出。第t时刻的输出门单元为：

$$g^{(t)} = \delta(U^o x^{(t)} + W^o h^{(t-1)} + b^o)$$

（摘录自课本 Page 88-89）

3.8 简述GAN的训练过程。

生成对抗网络(Generative Adversarial Net, GAN)分为两个部分，生成网络(NN Generator)和判别网络(Discriminator)。

在训练阶段，我们需要对生成网路和判别网络分别进行迭代训练，首先从数据集里sample出一小部分训练数据，然后fix住生成网络的参数，训练判别网络，使其对训练数据的输出越大越好、对生成网络生成的杂讯输出越小越好。接着fix住判别网络的参数，训练生成网络，让生成网络的输出经过判别网络之后的值越大越好，然后不断迭代训练。

3.9 简述图像风格迁移应用的基本过程。

首先输入内容图像p和风格图像a，图像风格迁移需要把二者的特征图结合起来。将内容图像和特征图像分别经过CNN生成各自的特征图，组成内容特征集P和风格特征集A。然后输入一张随机噪声图像x，x通过CNN生成的特征图构成内容特征集F和风格特征集G，然后由P、A、F、G计算目标损失函数。通过优化损失函数来调整图像x，使得内容特征集F与P接近、风格特征集G与A接近，经过多轮反复调整，可以使得中间图像在内容上与内容图像一致，在风格上与风格图像一致。*（摘录自课本 Page 96）*

4 第4章 编程框架的使用

4.1 请创建一个常量，在屏幕上输出"Hello, Tensorflow!"

```
import tensorflow as tf

print(tf.__version__) # tf 1.15.0

_print_str = tf.constant("Hello, Tensorflow!")
_tf_print = tf.print(_print_str)
with tf.Session() as sess:
    sess.run(_tf_print)
```

4.2 请实现两个数的加法，即计算A+B并输出，其中A是常量，B是占位符，数据类型自定。

```
import tensorflow as tf

print(tf.__version__) # tf 1.15.0

_print_str = tf.constant("Hello, Tensorflow!")
_tf_print = tf.print(_print_str)
with tf.Session() as sess:
    sess.run(_tf_print)
```

4.3 请实现一个矩阵乘法，数据类型和规模自定，并分别使用CPU和GPU执行。

```
import tensorflow as tf
import time

print(tf.__version__)  # tf 1.15.0

def _process(target):
    start = time.time()
    with tf.device(target):
        A = tf.Variable(tf.random.normal((4000,4000)), dtype=tf.float32)
        B = tf.Variable(tf.random.normal((4000,4000)), dtype=tf.float32)
        C = tf.multiply(A, B)
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        print(sess.run(C))
    print(target, "process time", time.time() - start, 's')

if __name__ == '__main__':
```

```
    _process('cpu')
    _process('gpu') if tf.test.is_gpu_available() else print("no gpu avaliable")
```

4.4 请重构本章中的build_vggnet()、read_wb()、basic_calc()函数，使得在构建网络过程中只需要打开一次权重文件。

没有提供vgg_model.npy文件(

仅需要打开一次权重文件，那么就在build_vggnet之前把权重文件读入成为params dict、之后就直接对params dict进行操作就好了。

4.5 请调研了解OpenCV、Skimage和PIL等图像处理库的层次接口。使用这些库读入图片数据，以默认参数读入常规彩色图片后，存储数据时在Channel纬度分别是RGB还是BGR的顺序？这在数据预处理时需要注意什么？

Code:https://github.com/LeiWang1999/AICS-Course/tree/master/Code/4.5.imagemodule.python

在默认参数的情况下，对于OpenCV，读入彩色图像时候是BGR顺序、而对于Skimage和PIL是RGB接口。

并且在利用这些接口进行数据预处理时，不仅要注意RGB通道的顺序差异，还需要注意的有：

1. 数据读入和存储的类型，例如使用PIL框架进行图像存储，仅能存储uint8类型的图像，而cv可以存储任意类型，在进行深度学习推理的时候，网络的输入有时候是归一化到0～1之间的数据，也可能是uint8的数据，如果不同则推理的结果会有较大差异。
2. 还需要注意通道差异，例如读入的顺序是h,w,c,但对于一些深度学习框架、读入图像的顺序是c,h,w，如果输入的数据格式不同，会影响推理的结果。

4.6 请调研了解常用的图像数据预处理和数据增强的方法。实现一个函数，从ImageNet2012_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，然后剧中裁剪为(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 * width, 0.875 * height, 3)大小的图片，再调整为（224，224，3）大小的图片。

4.6.1 常用的数据预处理方法：

- 进行图像的resize，因为神经网络的输入图像大小固定，所以需要resize。
- 每个图像通道减去均值mean，这一步的主要目的是消除图像的共性，突出每个图像的个性，也有的说法是将数据的分布调整到0中间、这样使用梯度下降反向传播算法的时候梯度的曲线就不会那么陡峭。

4.6.2 常见的数据增强方法

深度学习算法需要大量的训练数据，而有时我们收集不到太多的数据源，那么为了扩大数据集，可以采用数据增强手段来增加样本，常用的方法有：

- 随机裁剪
- 翻转或者镜像
- 旋转
- 调节亮度或者对比度

- 调节色度
- 调节图像的饱和度
- 将图像进行高斯模糊、锐化、添加噪声以及转换成灰度图像

要求编写的程序代码：https://github.com/LeiWang1999/AICS-Course/tree/master/Code/4.6.preprocess.python

4.7 在神经网络训练过程中有时需要使用动态学习率。已知初始化学习率为0.1。每进行10000次迭代，学习率变为之前的0.9倍，使用梯度下降优化器和其他API实现该需求。

可变学习率可以通过tensorflow的`train.exponential_decay`接口来实现：

```
starter_learning_rate = 0.1
steps_per_decay = 10000
decay_factor = 0.9
global_step = tf.Variable(0, trainable=False)
learning_rate = tf.train.exponential_decay(learning_rate =
starter_learning_rate,
                                           global_step = global_step,
                                           decay_steps = steps_per_decay,
                                           decay_rate = decay_factor,
                                           staircase = True,#If `True` decay
the learning rate at discrete intervals
                                           #staircase = False,change learning
rate at every step
                                           )

```

5 第5章 编程框架使用

5.1 请调研学习Eager API的使用。使用Eager API实现两个数的加法和矩阵乘法。

```python
import tensorflow as tf

print(tf.__version__)
tf.compat.v1.enable_eager_execution()
print(tf.executing_eagerly())

A = [[2.]]
B = [[3.]]
m = tf.matmul(A, B)
print(m)

n = tf.add(A, B)
print(n)
```

5.2 现有常见的编程框架的执行模式分为静态图模式和动态图模式，说明这两种执行模式各有什么优缺点。

静态图和动态图的区别在于，静态图需要我们事先定义好一张计算图并进行编译，之后在运行的过程中我们是对同一张计算图重复运算，不能更改计算图的内容。而动态图则在使用时创建。

静态图只需要编译一次，重复使用，这在部署上很实用，比如可以在磁盘中序列化，保存整个网络的结构，可以重载，而动态图则需要重复之前的代码。但编写静态图的程序需要使用特定的语法，增加了学习的成本，动态图可以直接使用Python语法，并且在调试过程中方便Debug。

5.3 使用GPU计算时，试分析在单机单卡、单机多卡、多机多卡的设备下训练卷积神经网络流程上的区别。其中哪些步骤是可以并行的，哪些步骤是必须串行的?

单GPU训练 一般代码比较简单，并且能满足我们的基本需求，通常做法是设定变量CUDA_VISIBLE_DEVICES的值为某一块GPU来Mask我们机器上的GPU设备，虽然有时当我们忘了设定该变量时程序会自动占用所有的GPU资源，但如果没有相应的代码去分配掌控GPU资源的使用的话，程序还是只会利用到第一张卡的计算资源，其他的资源则仅是占用浪费状态。

多GPU训练 则可以从两个方面提升我们模型训练的上限：1. 超过单卡显存上限的模型大小， 2. 更大的Batch Size和更快训练速度。相应的，目前各大主流框架的多GPU训练一般存在两种模式：

- **模型并行**： 分布式系统中的不同GPU负责网络模型的不同部分，进而可以 **构建超过单卡显存容量大小的模型** 。比如，可以将神经网络的不同层分配到不同的GPU设备，或者将不同的参数变量分配到不同的GPU设备。
- **数据并行**： 不同的 GPU设备有同一模型的多个副本，将数据分片并分配到每个GPU上，然后将所有GPU的计算结果按照某种方式合并，进而可以**增加训练数据的Batch Size**。

多机多卡相比较于单机多卡，其使得模型训练的上限进一步突破。一般我们一台服务器只支持8张GPU卡，而采用分布式的多机多卡训练方式，可以将几十甚至几百台服务器调度起来一起训练一个模型。

但相比于单机多卡，多机多卡分布式训练方式的配置更复杂一些，不仅要保证多台机器之间是可以互相通信的，还需要配置不同机器之间的角色以及不同机器之间梯度传递。

5.4 查看TensorFlow源码，在python/ops中，查找涉及注册sin算子梯度计算和maxpool算子梯度计算的代码，查看相关文件里注册其他算子的代码，学习了解注册Python层算子。

在`math_grad.py`里可以找到sin算子梯度计算的代码：

```
@ops.RegisterGradient("Sin")
def _SinGrad(op, grad):
  """Returns grad * cos(x)."""
  x = op.inputs[0]
  with ops.control_dependencies([grad]):
    x = math_ops.conj(x)
    return grad * math_ops.cos(x)
```

而在`nn_grad.py`里，有maxpool的梯度计算注册。

要整合自定义运算，您需要执行以下操作：

```
在 C++ 文件中注册新运算。运算注册会定义运算功能的接口（规范），此接口与运算的实现无关。例如，运算注册会定义运算的名称及运算的输入和输出，还会定义用于张量形状推断的形状函数。
使用 C++ 实现运算。运算的实现称为内核，它是您在第 1 步中注册的规范的具体实现。可以有多个内核用于不同的输入/输出类型或架构（例如，CPU、GPU）。
创建一个 Python 封装容器（可选）。此封装容器是用于以 Python 创建运算的公共 API。默认封装容器是根据运算注册生成的，用户可以直接使用它或向其中添加内容。
编写一个函数来计算运算的梯度（可选）。
测试运算。
```

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

在`nn_ops.cc`中有conv算子的注册流程，其中包括了一系列conv算子，例如Conv2D、Conv2DBackpropInput、FusedConv等等。

要整合自定义运算，您需要执行以下操作：

```
在 C++ 文件中注册新运算。运算注册会定义运算功能的接口（规范），此接口与运算的实现无关。例如，运算注册会定义运算的名称及运算的输入和输出，还会定义用于张量形状推断的形状函数。
使用 C++ 实现运算。运算的实现称为内核，它是您在第 1 步中注册的规范的具体实现。可以有多个内核用于不同的输入/输出类型或架构（例如，CPU、GPU）。
创建一个 Python 封装容器（可选）。此封装容器是用于以 Python 创建运算的公共 API。默认封装容器是根据运算注册生成的，用户可以直接使用它或向其中添加内容。
编写一个函数来计算运算的梯度（可选）。
测试运算。
```

6 第6章 深度学习处理器原理

6.1 深度学习处理器和20世纪80年代的早期神经网络芯片有何区别?

20世纪80年代早期的神经网络芯片只能处理很小规模的浅层神经网络算法，并且没有取得工业实践中的广泛应用。首先，当时的神经网络因为计算能力的限制没有发展到深层的神经网络，也没有针对多层神经网络的训练算法，加之当时的主流集成电路工艺还是1微米的工艺（当前主流集成电路的工艺已达到0.007微米），无法用有限的硬件神经元支持大规模的算法神经元。

而深度学习处理器不受神经网络规模限制，可以灵活、高效地处理上百层、千万神经元、上亿突触的各种深度学习神经网络（甚至更大）。

6.2 假设存在一个深度学习处理器，相较于CPU，它只能加速卷积层10倍，加速全连接层2倍，其他层不加速，那么对于AlexNet它整体的加速比是多少?

我们假设在ALexNet在CPU上，卷积层耗时t_c.全连接层耗时t_f。则在加速器上，卷积层耗时为$\frac{t_c}{10}$,全连接层耗时为$\frac{t_f}{2}$,则加速比为:

$$Rate = \frac{t_c + t_f}{\frac{t_c}{10} + \frac{t_f}{2}}$$

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?

最理想的情况下，一层全连接层刚好有12.8Gb的权重，进行全连接层运算的时候，12.8Gb的权重一次性从DDR3中读入，然后并行运算得到结果，而ALU只能一次读入一组数据进行运算，加速比为12.8G倍。

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?

指令集并行的优点在于灵活性高，缺点在于指令流水线的控制逻辑复杂，功耗、面积开销都很大。由于深度学习中主要是规整的向量、矩阵操作（尤其是卷积层和全连接层），行为可控，且深度学习处理去迫切需要提升效率，降低功耗及面积开销，因此深度学习处理器应尽可能简化控制通路，不适合采用复杂的流水线结构。

当深度学习算法的加速，涉及到复杂的控制逻辑的时候，指令集并行能够发挥作用。

7 第7章 深度学习处理器架构

7.1 本章所介绍的单核深度学习处理器的片内存储和传统CPU的片上缓存有什么区别?

GPT:

单核深度学习处理器的片内存储和传统CPU的片上缓存有一些区别。

首先，传统CPU的片上缓存主要是用于存储正在运行的程序的指令和数据，以便更快地访问。而单核深度学习处理器的片内存储主要是用于存储深度神经网络中的参数和中间结果，以便更快地进行计算。

其次，单核深度学习处理器的片内存储通常比传统CPU的片上缓存更大，因为深度神经网络需要存储大量的权重参数和中间结果，以支持模型的计算。相比之下，传统CPU的片上缓存通常比较小，因为它主要用于存储当前正在执行的程序的指令和数据。

另外，由于深度神经网络中的计算具有高度的数据局部性，即相邻的计算单元通常需要访问相同的权重参数和中间结果，所以单核深度学习处理器的片内存储通常也比传统CPU的片上缓存更加优化，以最大程度地利用数据局部性提高计算效率。

综上所述，虽然单核深度学习处理器的片内存储和传统CPU的片上缓存都是在处理器内部存储数据的高速缓存，但它们的设计和用途略有不同，以适应不同的计算任务和算法的需求。

7.2 本章所介绍的单核深度学习处理器的访存行为与传统的CPU有什么区别?

GPT

```
访问模式：深度神经网络中的计算通常具有很高的数据局部性，即相邻的计算单元通常需要访问相同的权重参数和中间结果。因此，单核深度学习处理器的访存行为通常更加优化，以最大程度地利用数据局部性提高计算效率。

数据类型：深度神经网络中的数据类型通常为浮点数，而传统的CPU还需要支持其他数据类型，如整数和字符等。因此，单核深度学习处理器的访存行为在数据类型支持和对不同数据类型的处理方面可能与传统CPU存在差异。

存储模式：深度神经网络通常需要处理大量的权重参数和中间结果，而这些数据的存储模式可能与传统的数据存储方式不同。例如，深度神经网络中的权重参数通常需要经过预处理后才能被存储，而传统的数据存储方式可能无法满足这种要求。因此，单核深度学习处理器的访存行为在数据存储模式方面也可能与传统CPU存在差异。
```

7.3 本章所介绍的但和深度学习处理器的指令译码过程与传统的CPU体系结构有什么区别?

GPT

以下是深度学习处理器的指令译码过程与传统CPU的一些区别：

```
指令集：传统的CPU支持广泛的指令集，包括算术逻辑指令、分支指令、访存指令等，而深度学习处理器的指令集通常较为简单，主要支持基本的向量操作和矩阵运算等。

指令译码：传统的CPU的指令译码过程通常包括多个阶段，需要解码指令的操作码、操作数寻址模式等信息，然后根据指令执行单元的类型进行译码。而深度学习处理器的指令译码过程通常更为简单，因为其指令集较为固定且不太复杂。

并行性：深度学习处理器的指令译码过程通常需要支持高度的并行性，以满足深度神经网络的计算需求。因此，深度学习处理器的指令译码过程通常会针对向量和矩阵运算等计算密集型操作进行优化，以最大化处理器的并行计算能力。

特殊指令：由于深度神经网络的计算涉及到大量的矩阵运算和向量操作等，因此深度学习处理器的指令集通常会包括一些特殊的指令，以支持这些操作的高效执行。例如，一些深度学习处理器可能支持卷积运算、池化运算等特殊的指令，以加速卷积神经网络等模型的计算。
```

7.4 怎样改进才能使本章所介绍的单核深度学习处理器能够支持乱序执行的功能？从应用的角度考虑，深度学习处理器是否有必要支持乱系执行功能?

GPT

要支持乱序执行，需要在单核深度学习处理器的设计中引入乱序执行单元，该单元可以根据指令的相关性，自动重排指令的执行顺序，以充分利用处理器的资源，并提高计算效率。乱序执行需要考虑多个方面，包括指令调度、数据依赖性、控制依赖性、内存依赖性等。

从应用的角度考虑，乱序执行对于深度学习处理器的性能提升是非常重要的。深度学习模型的训练和推断通常需要大量的矩阵乘法、卷积运算和向量操作等密集型计算，这些操作之间存在着很强的并行性和重叠性。如果深度学习处理器无法支持乱序执行，将会限制处理器的并行计算能力和计算效率，导致处理器的性能瓶颈。

7.5 7.6 请简述DMA的工作过程及其关键要点。

书P208P215

GPT

DLP-S的DMA工作过程如下：

```
主机CPU设置DLP-S的DMA控制器：主机CPU通过PCIe总线将DMA控制器的寄存器设置为需要传输的数据大小、传输方向、源地址和目的地址等参数。

DLP-S的DMA控制器启动传输：DLP-S的DMA控制器根据CPU设置的参数，启动数据传输。它会向主机发出请求，申请PCIe总线的控制权，并在获得控制权后，将数据从主机内存传输到DLP-S的内存，或从DLP-S的内存传输到主机内存。

DLP-S的DMA控制器完成传输：当数据传输完成后，DLP-S的DMA控制器将释放PCIe总线控制权，并向主机发出中断请求，通知主机传输已经完成。
```

关键要点如下：

```
DMA控制器：DLP-S的DMA控制器是DMA传输的核心，负责控制数据传输的各个环节，包括请求PCIe总线控制权、获取数据、传输数据、释放PCIe总线控制权等。

数据传输方向：DLP-S的DMA支持两种传输方向，分别是从主机内存到DLP-S的内存（DMA读取）和从DLP-S的内存到主机内存（DMA写入）。

DMA优化：DLP-S的DMA控制器还支持多个DMA通道，可以同时进行多个数据传输任务，从而进一步提高数据传输效率。此外，DLP-S还支持DMA突发模式（burst mode），可以在一次请求中传输多个数据块，减少DMA控制器与PCIe总线之间的通信次数，进一步提高数据传输效率。
```

7.6　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，该芯片运行频率位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。

8　第8章 智能编程语言

8.1　假设某个处理器的存储单元包括一块片上高速存储和一块片外DDR，访问时间分别为4个时钟周期和150个时钟周期。如果工作负载在片上缓存的命中率为90%，而且处理器只有在片上缓存未命中的情况下才会访问片外DDR。那么，整个存储层次的平均访问延迟是多少个时钟周期？

GPT

根据问题描述，可以使用加权平均延迟（Weighted Average Access Time, WAAT）来计算整个存储层次的平均访问延迟。WAAT 的计算公式为：

$$WAAT = Hit\ time + Miss\ rate \times Miss\ penalty$$

其中，Hit time 是指命中时的访问时间，Miss rate 是未命中率，Miss penalty 是未命中时的额外开销。

根据问题描述，片上缓存的命中率为90%，所以未命中率为10%。根据加权平均延迟的公式，可以计算出整个存储层次的平均访问延迟：

$$WAAT = 0.9 \times 4 + 0.1 \times 150 = 7.6$$

因此，整个存储层次的平均访问延迟为7.6个时钟周期。

8.2　在异构编程模型的处理器端流程中，介于创建Stream和启动Kernel之间的两个流程步骤是什么？

GPT

1.创建Stream：在处理器端创建一个新的Stream对象，用于将需要在设备上执行的Kernel函数加入到队列中，等待设备的执行。

2.设置Kernel参数：调用处理器端的API函数，将Kernel函数需要的参数传递给设备。这些参数通常包括输入数据、输出数据、数据大小和数据类型等。

3.复制数据到设备：将需要在设备上使用的数据从主机内存中复制到设备内存中，以便Kernel函数可以访问这些数据。这个过程通常涉及到两个操作：将数据从主机内存中复制到设备端的内存缓存中，然后再将数据从设备端的内存缓存中复制到设备端的全局内存中。

4.启动Kernel：将Kernel函数添加到处理器端的队列中，等待设备执行。一旦Kernel函数被添加到队列中，它将在设备端的处理器上执行，并将其计算结果写回到设备端的内存中。

8.3 假设有一个8位的二进制数是10011001，如果它代表无符号整数uint8，那么它换算成十进制数是多少？如果他代表有符号整数int8，那么它换算成十进制数是多少？

GPT

12^7 + 02^6 + 02^5 + 12^4 + 12^3 + 02^2 + 02^1 + 12^0 = 128 + 16 + 8 + 1 = 153

因此，10011001二进制数代表的无符号整数uint8为153。

对于有符号整数int8，给定的8位二进制数10011001表示的是负数。首先需要将其转化为补码表示形式，即将该数的绝对值取反后加1。该操作的步骤如下：

取反得到01100110；加1得到01100111。

因此，10011001二进制数代表的有符号整数int8为-103。

8.4 假设有一个32位的浮点数，其二进制形式是1 1000 0101 1111 1110 0000 0000 0000 000，如果按照IEEE754标准，换算为十进制整数是多少？如果二进制是1 0000 0000 1000 0000 0000 0000 0000 000，那么换算位十进制数是多少？

GPT

符号位 s = 1，表示负数；
指数位 eeeeeeee = 10000101_2 - 127_{10} = -22_{10}，表示指数为负数；
尾数位 mmmmmmmmmmmmmmmmmmmmmmm = $1.11111111000000000000000_2$。

因此，根据IEEE754标准的公式：(-1)^(s) * (1 + m) * 2^(e)，可以得到该浮点数的十进制表示为：-1.00002441406251_{10}。

对于第二个问题，给定的32位浮点数的二进制形式为：10000000010000000000000000000000，可以将其分为三个部分：

符号位 s = 1，表示负数；
指数位 eeeeeeee = 00000000_2 - 127_{10} = -127_{10}，表示指数为负数；
尾数位 mmmmmmmmmmmmmmmmmmmmmmm = $1.00000000000000000000000_2$。

因此，根据IEEE754标准的公式：(-1)^(s) * (1 + m) * 2^(e)，可以得到该浮点数的十进制表示为：-2.