中国科学技术大学

博士学位论文

中国科学技术大学

学位论文模板示例文档

作者姓名: 李泽平

学科专业: 数学与应用数学

导师姓名: 华罗庚 教授　钱学森 教授

完成时间: 二〇一八年六月二十一日

University of Science and Technology of China

A dissertation for doctor's degree

An Example of USTC Thesis Template for Bachelor, Master and Doctor

Author: Zeping Li

Speciality: Mathematics and Applied Mathematics

Supervisors: Prof. Luogeng Hua, Prof. Xuesen Qian

Finished time: June 21, 2018

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□ 公开　□ 保密（____ 年）

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘　要

摘要是论文内容的总结概括，应简要说明论文的研究目的、基本研究内容、研究方法或过程、结果和结论，突出论文的创新之处。摘要中不宜使用公式、图表，不引用文献。博士论文中文摘要一般 800 ~ 1000 个汉字，硕士论文中文摘要一般 600 个汉字。英文摘要的篇幅参照中文摘要。

关键词另起一行并隔写在摘要下方，一般 3 ~ 8 个词，中文关键词间空一字或用分号“;”隔开。英文摘要的关键词与中文摘要的关键词应完全一致，中间用逗号“,”或分号“;”隔开。

关键词：中国科学技术大学；学位论文；LaTeX 模板；学士；硕士；博士

ABSTRACT

This is a sample document of USTC thesis LaTeX template for bachelor, master and doctor. The template is created by zepinglee and seisman, which orignate from the template created by ywg. The template meets the equirements of USTC theiss writing standards.

This document will show the usage of basic commands provided by LaTeX and some features provided by the template. For more information, please refer to the template document ustcthesis.pdf.

Key Words: University of Science and Technology of China (USTC); Thesis; LaTeX Template; Bachelor; Master; PhD

目　录

第 1 章　简　　介

1.1　一级节标题

1.1.1　二级节标题

1. 三级节标题

（1） 四级节标题

① 五级节标题

1.2　脚注

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. [①]

[①] This is a long long long long long long long long long long long long long long long long long long long long long long long long footnote.

第 2 章　神经网络简介

2.1　神经网络算法基础

现代神经网络由多种不同类型的层组成，输入数据依次通过各个层被逐层处理，最终被分类, 识别或者检测。在每一层中，神经元接收多个输入进行处理，然后通过连接将输出发送到下一层。神经元之间的连接，即所谓的突触，通常具有独立或者共享的权重。神经网络在图像处理，语音识别，语音合成，自然语言处理等领域有着非常广泛的应用。目前在图像处理应用最广泛的网络是卷积神经网络（convolutional neural networks, 简称 CNNs）和深度神经网络（deep neural networks, 简称 DNNs）, 它们由卷积层，池化层，归一化层，激活层和全连接层等组成。递归神经网络（recurrent neural networks，简称 RNNs）是一类重要的机器学习技术，专门用于处理顺序数据序列和可变长度数据序列。RNNs 在语音识别，自然语言处理，场景语义理解和时间序列分析等方面有着广泛的应用。其中应用最广泛，性能最高的两种类型的 RNNs 是长短时记忆网络（long short term memory, 简称 LSTM）和门控循环单元 (gated recurrent unit, 简称 GRU)。下面将为大家介绍神经网络中集中常见的神经网络层类型。

2.1.1　全连接层

全连接层（fully connected layers）是神经网络算法中常见的一种层类型，主要用来将上一层提取的特征进行组合，综合以及分类，在整个神经网络中起到“分类器” 的作用。全连接层的结构如图 2.1所示，输入层神经元为 m 个，输出层的神经元为 n 个，其中每一个输出神经元与所有输入神经元相连，其结构相当于 n 个 m 输入的感知机，因此全连接层也可以看成是感知机扩展。

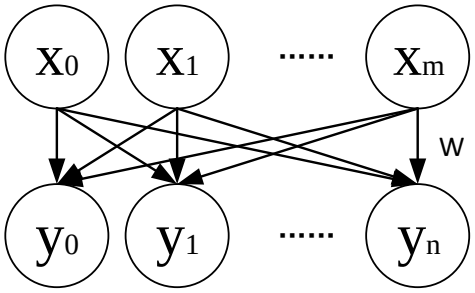

图 2.1　全连接层.

全连接层的核心操作操作是矩阵向量乘积，

$$y = W * x + b \tag{2.1}$$

其中 y 是输出神经元向量，x 是输入神经元向量，W 是权值矩阵，b 是输出神经元的偏置向量。因此全连接层的本质是由一个特征空间线性变换到另一个特征空间，且目标空间的任一维都会收到源空间每一维的影响。由于每个输出神经元都要与输入神经元相连接，这样的话就会造成权值数量巨大，从而造成网络难以训练，并且会出现过拟合的情况。因此在 CNN 中，全连接层常出现在最后几层，用取于对前面设计提的特征做加权和。

2.1.2　卷积层

卷积层（convolutional layers）是卷积神经网络的核心层，主要用于提取特征。卷积层受到生物学上感受野（Receptive Field）的机制而提出的。感受野主要是指听觉系统、本体感觉系统和视觉系统中神经元的一些性质。比如在视觉神经系统中，一个神经元的感受野是指视网膜上的特定区域，只有这个区域内的刺激才能够激活该神经元。在听觉系统中，对于语音则是某一时间戳后的时间段才能激活神经元。卷积层充分借鉴感受野的机制，神经元仅仅能够感受局部特征，而卷积核的大小直接限制感受野的大小，输出神经元只和周围一小块的输入神

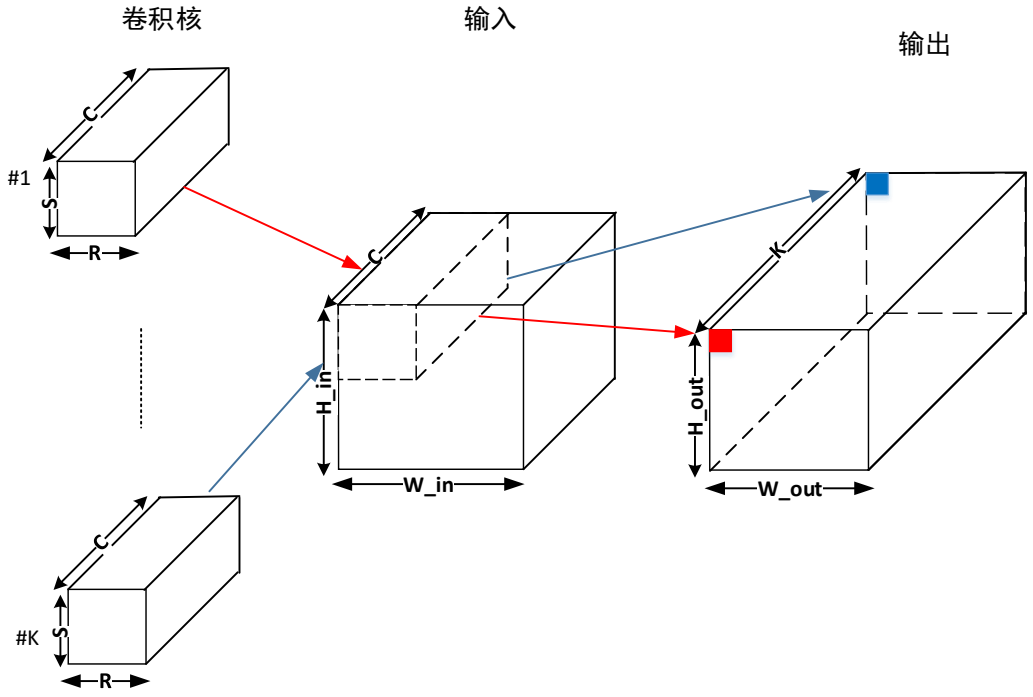

图 2.2　卷积层.

经元存在连接。局部连接的方法大大减少了卷积层中的权值数量。卷积层采用了共享权值的方法进一步减少权值的数量，即同一输出特征图共享一组卷积核。同一个输出特征图像上的所有像素点是同一组卷积核在输入图像不同位置上提取到的图像特征, 而这些图像特征具有相同意义。卷积层采用多组不同的卷积核对输入特征图进行卷积，从而获得多组输出特征图，即多组不同的图像特征。

卷积层的结构如图 2.2所示。卷积层的核心运算是二维滑动窗口卷积运算，即规模为 $R \times S$ 的卷积核在规模为 $W_{in} \times H_{in}$ 的输入特征图上滑动进行二维卷积，最终产生规模为 $W_{out} \times H_{out}$ 的输出特征图。通常情况下，输入特征图不仅只有一个，而是由 C 个组成，即规模为 $C \times W_{in} \times H_{in}$，因此我们对每个输入通道都施加一个卷积核，即使用规模为 $C \times R \times S$ 的卷积核对输入进行卷积，最终获得一个输出特征图。当我们将采 K 组不同的卷积核作用于相同的输入特征图，将获得 K 个不同的输出特征图。最终，卷积层的输入规模为 $C \times W_{in} \times H_{in}$，卷积核的规模为 $K \times C \times R \times S$, 输出规模为 $K \times W_{out} \times W_{out}$。

```
1  for k = 0 to K do
2      for w = 0 to W_out do
3          for h = 0 to H_out do
4              for c = 0 to C do
5                  for s = 0 to S do
6                      for r = 0 to R do
7                          out[k][w][h]+ =
                             in[c][w * sw + r][h * sh + s] * filter[k][c][r][s]
8                      end
9                  end
10             end
11         end
12     end
13 end
```

算法 2.1: 六层卷积循环

卷积层的计算由 K，C，R，S，W 和 H 这六个变量形成嵌套循环完成，而且这六个变量的所有排列都是合法的。算法 2.1展示了其中一种循环嵌套方式，我们可以用 $N->W_{out}->H_{out}->C->S->R$ 来描述这种循环，其中 sw 和 sh 表示卷积操作的步长。不同的循环方式决定了数据的复用形式和数据流的方式，最终将影响神经网络加速器的设计。

2.1.3　池化层

池化层（pooling layer）是神经网络中一个重要的层。池化层一般是在卷积层之后，对输入进行非线性降采样，常用的池化做法是对每个滤波器的输出求最大值，平均值，中位数等。池化层的意义主要体现在两个方面：第一，池化层通过对特征图像进行降维操作，能够在保留显著特征的情况下，有效减少整个神经网络所需要的参数量和计算量。第二，池化层能够保证输入的平移不变性(translation invariant)，这意味着即使图像的像素在邻域发生微小位移时，池化层的输出能够保持不变，从而增强神经网络的鲁棒性，由一定的抗扰动能力。常用的池化层包括最大池化层，平均池化层，ROI (Regions of interest) 池化层.

最大池化层的基本思想是在一个特定的数据区域内选择一个最大值作为输出。其计算公式是

$$out_{x,y} = max(in_{x*sx,y*sy} : in_{x*sx+kx,y*sy+ky}) \tag{2.2}$$

其中 kx 和 ky 是池化窗口的大小，sx 和 sy 是池化的步长。通常情况下池化窗口的大小与池化的步长相同，即池化操作的输入并不会重复。后来也出现了数据复用的池化，相邻池化窗口之间会有重叠区域，此时池化步长小于池化窗口的大小。

平均池化层的基本操作与最大池化层类似，它将一个特定的区域内的数据取算数平均作为输出，其计算公式是

$$out_{x,y} = mean(in_{x*sx,y*sy} : in_{x*sx+kx,y*sy+ky}) \tag{2.3}$$

ROI Pooling 层主要是针对 ROIs 的池化操作，主要应用于物体检测领域的 Fast RCNN 和 Faster RCNN 网络中，它的特点是输入特征图尺寸不固定，但是输出特征图的尺寸固定。ROI Pooling 的输入包含两部分，第一部分是特征图，在 Fast RCNN 中，它位于 ROI Pooling 之前；在 Faster RCNN 中，它是与 RPN 共享的那个特征图。第二部分是 ROIS，在 Fast RCNN 中，指的是 Selective Search 的输出；在 Faster RCNN 中指的是 RPN 的输出，是一系列的矩阵候选框，每一个矩阵候选框用四个坐标和索引来表示。ROI Pooling 的输出则是 batch 个三维向量（$C \times W \times H$），其中 batch 的值为 ROI 的数量。因此 ROI Pooling 的过程可以总结为将大小不同的矩阵框映射为大小固定的矩阵框。

2.1.4　归一化层

随着神经网络的规模不断变大，结构的复杂度增加，神经网络越来越难以训练，同时神经网络越来越容易出现过拟合的现象。归一化层（normalization layer）成为神经网络中不可或缺的一个层，它能够加快神经网络的收敛速度，防止过拟

合,降低神经网络对初始化权重的敏感度,提高神经网络的精度。近几年出现了各种各样的归一化方法, 主要包括 LRN(Local Response Normalization) [1],BN(Batch Normalization) [2]，LN(layer Normalization) [3]，IN(Instance Normalization) [4] 和 GN(Group Normalization) [5] 等。

LRN 是 AlexNet 等网络中使用的归一化方法，它在每一个像素的小领域范围内进行归一化处理。目前主流的归一化方法则更加注重全局范围内的归一化处理，这些全局归一化的方法能够使得我们在训练神经网络时使用较大的学习率，从而加快神经网络训练速度。如图 2.3所示，BN 选择在 batch 维度上进行归一化处理；LN 选择在 channel 维度进行归一化处理；IN 执行类似 BN 的计算, 但是仅仅在单个样本执行归一化；GN 在 IN 的基础上对多个样本进行归一化。下面主要对 BN 的计算方法进行简要介绍。

BN 的计算公式如下所示：

$$\mu_\beta = \frac{1}{N}\sum_{i=1}^{N} a_i \tag{2.4}$$

$$\theta_\beta^2 = \frac{1}{N}\sum_{i=1}^{N}(a_i - \mu_\beta)^2 \tag{2.5}$$

$$\hat{a}_i = \frac{a_i - \mu_\beta}{\sqrt{\theta_\beta^2 + \epsilon}} \tag{2.6}$$

$$b_i = \gamma \hat{a}_i + \beta \tag{2.7}$$

在上述算法中，m 为 batch 数，a_i 是第 i 个 batch 的输入数据，μ_β 和 θ_β^2 分别是 N 个输入的均值和方差，参数 ϵ 是批变化常量，参数 γ 和 β 是训练时需要学习的参数，b_i 是最后归一化后的输出。

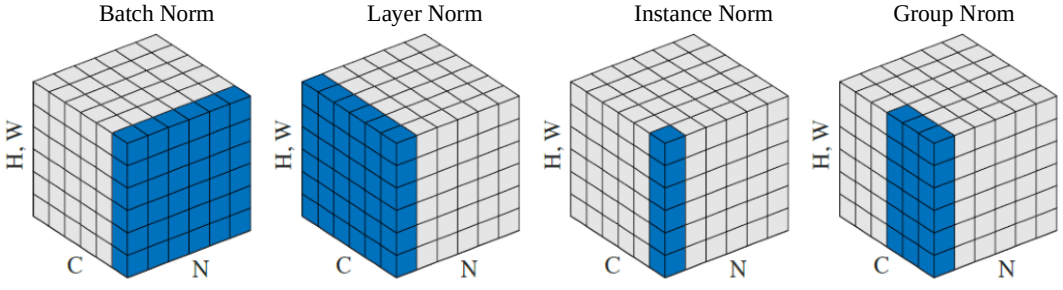

图 2.3　归一化方法。每一个子图显示的是一个 feature map，其中 N 为 batch 轴，C 为 channel 轴，（H,W）为空间轴。蓝色的像素表示归一化的范围。

2.1.5　激活层

激活层（activation layer）是神经网络能够解决非线性问题的关键，它弥补了神经网络中线性模型表达能力不足。目前主流的激活函数包括 Sigmoid, Tanh, ReLU 以及 ReLU 的变种, 如 PReLU 和 RReLU 等。

Sigmoid，Tanh 和 ReLU 的函数图如图 x 所示，公式分别为

$$Sigmoid(x) = \frac{1}{1+\mathrm{e}^{-x}} \tag{2.8}$$

$$Tanh(x) = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}} \tag{2.9}$$

$$ReLU(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{if } x \leqslant 0 \end{cases} \tag{2.10}$$

Sigmoid 函数主要被应用于早期的神经网络中,Sigmoid 函数拥有良好的性质，Sigmoid 函数的值域范围限制在 (0,1) 之间，这与概率值的范围是相对应的，这样 sigmoid 函数就能与一个概率分布联系起来了; 同时 Sigmoid 函数的导数可以由其本身求得 $Sigmoid(x)' = Sigmoid(x)(1 - Sigmoid(x))$。但是 Sigmoid 函数也有不少缺点，首先 Sigmoid 函数的输出并不是以 0 为中心，这个特性会导致为在后面神经网络的高层处理中收到不是零中心的数据，进而训练时的权值更新产生锯齿晃动。同时 Sigmoid 函数具有饱和性，容易出现梯度消失的现象，使得神经网络难以训练。

Tanh 函数是 Sigmoid 函数的改进版本，它的收敛速度比 Sigmoid 函数更快，相比 Sigmoid 函数，其输出以 0 为中心, 但是仍然存在饱和性而导致梯度消失的问题的。Tanh 函数和 Sigmoid 函数目前主要被用于基于 LSTM 的 RNN 架构或者基于 GRU 的 RNN 架构中。

ReLU 是目前非常流行的激活函数，它是分段线性函数，所有的负值为 0，而正值不变，这种操作被称为单侧抑制。单侧抑制使得神经网络中的神经元也具有了稀疏激活性，尤其体现在深度神经网络模型 (如 CNN) 中，当模型增加 N 层之后，理论上 ReLU 神经元的激活率将降低 2 的 N 次方倍，从而更好地挖掘相关特征，拟合训练数据。同时，ReLU 不存在饱和区，因此不存在梯度消失的问题，使得模型的收敛速度维持在一个稳定的状态。[1] 实验显示 ReLU 单元比 tanh 单元具有 6 倍的收敛速度提升。

2.1.6　LSTM

现代大规模自动语音识别（automatic speech recognition，简称 ASR）系统利用基于 LSTM 的 RNN 作为其声学模型。LSTM 模型由一系列大规模矩阵组成，这是 ASR 的所有步骤中计算量最大的部分。在基于 LSTM 的 RNN 中，时刻 T

的输入取决于时刻 T-1 的输出。一个经典的 LSTM 模型如图 2.4所示。LSTM 模型包含特殊存储单元（图 2.4中的 $cell$）和三个特殊的门（图 2.4中的 i，o，f），其中 $cell$ 用于存储网络的时间状态信息，门用于执行特殊的乘法运算，包括输入门（input gate）、输出门（output gate）和遗忘门（forget gate）。输入门 i 控制输入到存储单元中的输入量；输出门 o 控制输出值；遗忘门 f 自适应地遗忘 $cell$ 中存储的信息，从而控制前一状态对现状态的影响。除了基本的三个众所周知的门和 $cell$ 之外，该 LSTM 模型还引入了窥视孔（peephole）和投影层（projection layer），以便更好地学习。窥视孔将缩放后的 $cell$ 状态添加到三个门，其中缩放尺度由三个对角矩阵决定。投影层线性地将输出转换为低维形式。

LSTM 模型的接收一个输入序列 $X = (x_1; x_2; x_3; ...; x_T)$（其中 x_t 是 t 时刻的输入向量）和上一个状态的输出序列 $Y^{T-1} = (y_0; y_1; y_2; ...; y_{T-1})i$（其中 y_{t-1} 是 t-1 时刻的输出向量）。利用下列公式从 t=1 到 T 计算输出序列 $Y = (y1; y2; y3; ...; y_T)$:

$$i_t = \delta(W_{ix}x_t + W_{ir}y_{t-1} + W_{ic}c_{t-1} + b_i), \tag{2.11}$$

$$i_t = \delta(W_{ix}x_t + W_{ir}y_{t-1} + W_{ic}c_{t-1} + b_i), \tag{2.12}$$

$$f_t = \delta(W_{fx}x_t + W_{fr}y_{t-1} + W_{fc}c_{t-1} + b_f), \tag{2.13}$$

$$g_t = \delta(W_{cx}x_t + W_{cr}y_{t-1} + b_c), \tag{2.14}$$

$$c_t = f_t \odot c_{t-1} + g_t \odot i_t, \tag{2.15}$$

$$o_t = \delta(W_{ox}x_t + W_{or}y_{t-1} + W_{oc}c_t + b_o), \tag{2.16}$$

$$m_t = o_t \odot h(c_t), \tag{2.17}$$

$$y_t = W_{ym}m_t \tag{2.18}$$

其中符号 i、f、o、c、m 和 y 分别是输入门、遗忘门、输出门、$cell$ 状态、$cell$ 输出和投影输出；$\odot$ 表示向量逐元素乘积，W 表示权值矩阵（例如 W_ix 是从输入向量 X_t 到输入门的权值矩阵），b 表示偏置向量。值得注意的是 W_ic、W_fc 和 W_oc 是用于 peephole 连接的对角矩阵，因此它们本质上是向量。δ 是 Sigmoid

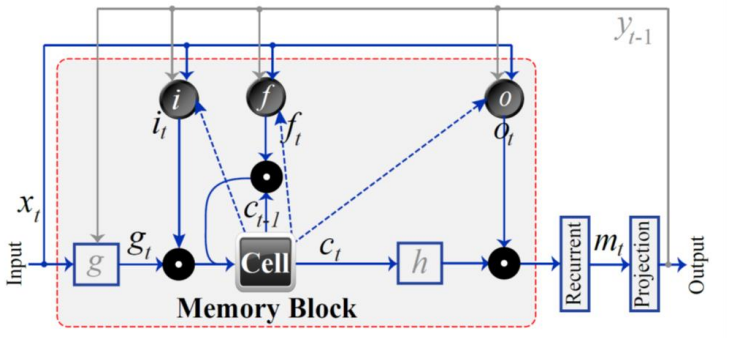

图 2.4　一个基于 LSTM 的 RNN 架构.

激活函数，h 是用户自定义的激活函数，通常情况下会采用 $tanh$ 激活函数。

2.1.7　GRU

GRU 是 LSTM 的变体，它把遗忘门和输入门组合成一个更新门（update gate）。它还合并了 $cell$ 状态和隐藏状态，并进行了一些其他更改。GRU 架构如图 2.5所示。类似地，它遵循如下的公式从 t = 1 到 T 进行迭代计算：

$$z_t = \delta(W_{zx}x_t + W_{zc}c_{t-1} + b_z), \tag{2.19}$$

$$r_t = \delta(W_{rx}x_t + W_{rc}c_{t-1} + b_r), \tag{2.20}$$

$$\tilde{c_t} = h(W_{\tilde{c}x}x_t + W_{\tilde{c}c}(r_t \odot c_{t-1}) + b_{\tilde{c}}), \tag{2.21}$$

$$c_t = (1 - z_t) \odot c_{t-1} + z_t \odot \tilde{c_t} \tag{2.22}$$

其中符号 z、r、$\tilde{c}$、c 分别是更新门、复位门、复位状态和 $cell$ 状态；$\odot$ 表示逐元素乘法。W 表示权值矩阵，δ 是 Sigmoid 激活函数，h 是用户定义的激活函数，这里我们使用 $tanh$ 激活函数。值得注意的是 GRU 有两个门（更新门和复位门），而 LSTM 有三个门（输入门、遗忘门、输出门），GRU 中不存在 LSTM 中存在的输出门，而是将 $cell$ 状态作为输出。LSTM 中输入门和遗忘门耦合称为 GRU 中的更新门 z，复位门 r 直接作用于到上一个 $cell$ 的状态。

2.2　神经网络低能耗的技术

随着神经网络算法被应用于更复杂的处理任务和更广泛的场景中，神经网络的规模也越来越大。最新的 DNNs [] 需要数百兆字节甚至前兆字节来存储神经元和权值；同时需要数十亿次的乘加操作完成运算。考虑未来神经网络将朝着

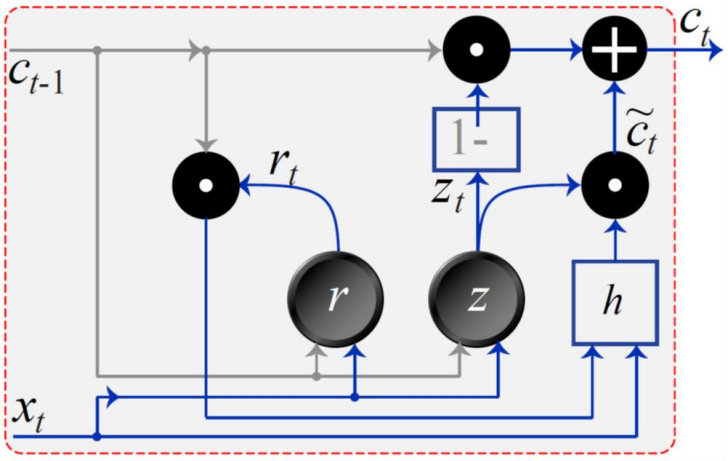

图 2.5　一个基于 GRU 的 RNN 架构.

规模更大，层数更深的趋势发展，未来的大规模网络很难部署到嵌入式系统中。因此很多研究人员致力于减少神经网络运算需求，降低神经网络运算时的能耗。目前已经有了一些有效的算法来解决这个问题，主要方法包括神经网络的低精度计算和神经网络压缩模型。

2.2.1　神经网络的低精度计算

深度学习追求小的泛化误差，即能够在训练数据集外的新数据上获得好的性能。研究显示，非精确的计算（在神经元和权重中加入噪音）具有正则化的效果，从而减少泛化误差 [6]。低精度计算的方法包括定点数据格式，半精度格式，混合精度，二值化等。

]使用 8 比特或者 16 比特定点数据来表示权值，但是不影响神经网络的精度。]发现低精度数据不仅足以支撑神经网络预测阶段，还足以支撑训练。例如，利用 10 比特的乘法足以训练 MaxOut 网络。 Köster et al. [7]提出了动态数据格式（FlexPoint），用来完全替代 32 位浮点格式。Flexpoint 具有一个可动态调整的共享指数，以最小化溢出和最大化可用动态范围。在训练过程中，数据范围会随着训练轮数的增长而连续变化，其中共享指数部分可以根据历史数据进行预测。实验结果显示，采用 16 位尾数和 5 位共享指数的 Flexpoint 格式数据在训练神经网络时，能够获得 32 位浮点类似的精度，并且远远超过 16 位浮点获得的精度。

二进制网络（Binary Neural Networks，BNNs）是一种极限低精度神经网络模型。 Courbariaux et al. [8]提出了二进制权值，权重被限定为两个可能的值（例如-1 或 1），这种情况下标准神经网络中的乘法操作可以被简单的加法或者减法替代。因此神经网络处理器中的所有乘法器替换成加法器，从而大大减少处理器的面积，降低处理器的能耗。在二进制权值的基础上， Courbariaux et al. [9]将神经元和权值同时进行二值化，从而提出了完整了的 BNN。BNN 网络中，神经网络的所有运算都能用位运算来替代，从而大大降低神经网络的规模，加快神经网络的运算速度。然而 BNN 会严重降低神经网络的精度， Rastegari et al. [10]提出了 Xnor-net，对二进制网络的训练过程进行优化，解决了二进制数据表示范围狭窄的问题，提高网络的识别精度。

2.2.2　神经网络压缩模型

大规模的神经网络通常是过拟合的，过量的参数会严重影响神经网络的运算速度。因此，我们可以通过压缩神经网络模型的方式来缓解过拟合的情况，同时减少神经网络的存储需求和运算需求。神经网络压缩的方法可以分为两类，剪枝和矩阵分解。

[11] 提出了剪枝的策略来减少神经网络中权值的数量，在不影响神经网络准确性的前提下，能够将神经网络所需要的存储量和计算量减少一个数量级。剪枝策略首先对神经网络进行训练，找出那些重要的连接；接下来，修剪那些不必要的连接；最后对神经网络进行冲训练，微调剩余连接的权值。实验显示，剪枝策略能够将 AlexNet 网络的连接数量减少 9 倍，将 VGG16 网络的连接数量减少 13 倍。

在此基础上，Han et al. [12]进一步提出了 Deep Compression 用来深度压缩神经网络。Deep Comression 由三个阶段组成：剪枝（Pruning），量化（quantization）和霍夫曼编码（Huffman Coding），最终在不影响神经网络的精度的前提下将神经网络压缩了 35 倍到 49 倍。Deep Compression 首先通过剪枝阶段学习网络中重要的连接，删除不必要的连接，这个步骤能够减少 9 倍到 13 倍的权值。然后通过聚类算法将权值进行聚类，然后量化权值实现权值的共享，这个步骤能够减少表示每个权值的比特数，从 32 比特减少到到 5 比特。最后采用霍夫曼编码进一步无损压缩神经网络，这个步骤能够节省 20% 30% 的网络存储开销。在 ImageNet 数据集上，Deep Compression 能够将 AlexNet 网络压缩 35 倍，将网络规模从 $240MB$ 压缩到 $6.9MB$。同时，Deep Compression 将 VGG16 网络压缩了 49 倍，将网络规模从 552MB 压缩到 11.3MB。这将允许我们将网络模型存储在加速器的片上 SRAM 缓存中，而不需要在片上和片外之间反复搬运权值，从而减少片外访存开销。

Wang et al. [13]提出了一种新的有效的 CNN 压缩方法 CNNpack，它在频域对神经网络进行剪枝操作，因此它不仅能够关注小的权值，同时关注所有潜在的对计算结果影响小的连接。CNNpack 借助离散余弦变换（Discrete Cosine Transform，DCT）将空间域的权值变换到频域，然后采用聚类的方法将频域中的权重分解为公共部分和私有部分（残差）。在这两部分中，采用剪枝的策略丢弃大量的低能级的频率系数，能够在不显著降低精度的情况下产生高压缩比。在此基础上，CNNpack 接着采用量化，霍夫曼编码和 CSR 存储的方法进一步压缩神经网络。最终实验实验显示，CNNpack 能够对 AlexNet 和 VGG16 分别压缩 35 倍和 49 倍。

除了以上对权值进行剪枝的策略，还有不少研究工作直接对神经元进行剪枝操作 [14-17]。但是对神经元直接进行剪枝操作会严重降低神经网络的精度，因此不能获得高的稀疏度。Data-free Parameter Pruning [15] 在 AlexNet 和 LeNet5 上仅仅能够获得 65.11% 和 16.5% 的稀疏度，远远高于权值剪枝策略 [11] 的 11% 和 8%。Network Trimming [16] 能够在 VGG16 和 LeNet5 上获得 37% 和 26% 的系数率，也高于 [11] 中的 7.5% 和 8%。

除了经典的剪枝策略（包括权值剪枝和神经元剪枝），还有一部分的研究工作基于矩阵分解和因式分解 [18-20]，这些方法可以保持原始模型的规则密集计

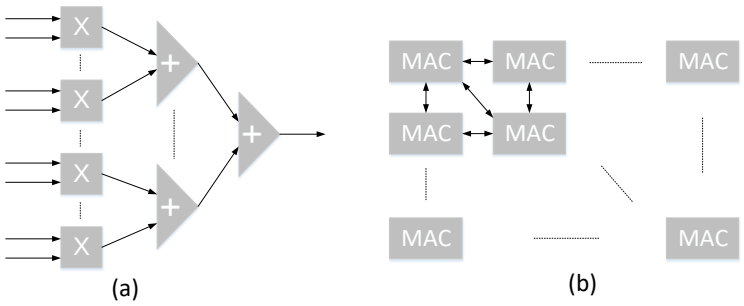

图 2.6　现有的加速器架构.

算结构，从而在通用处理器上实现压缩和加速。 Denton et al. [18]是利用神经网路线性结构的特性，对神经网络进行适当的低秩分解（low-rank approximation），在精度损失在 1% 的情况下，在很多模型上都能获得超过 2 倍的加速。 Jaderberg et al. [19]利用将 $k \times k$ 的卷积核分解为 $k \times 1$ 和 $1 \times k$ 的卷积核，从而加速卷积计算。同时 [19] 提出了两种优化方案：一种是利用滤波重构的方法最小化滤波权值误差，另一种是利用数据重构最小化响应误差的。 Lebedev et al. [20]则采用 CP decomposition 的方式对大规模神经网络的卷积核进行分解和加速，实验结果显示，在精度损失在 1% 的情况下，对 AlexNet 网络能够获得 4 倍的加速效果。

2.3　神经网络加速器

2.3.1　现有神经网络加速器架构

由于严峻的能耗约束和高性能要求，定制加速器成为 CPU 和 GPU 等传统处理平台的替代品。近几年出现了数据流和结构各异的神经网络加速器。神经网络加速器按照数据流的形式可以分为两类，分别是基于向量算子数据流的加速器和和基于乘加算子（multiply-and-accumulate，简称 MAC）空间数据流的加速器。基于向量算子的加速器将神经网络的运算转化为向量运算，主要是将矩阵操作转化为一系列的向量操作（通常是内积操作）。如图 2.6 (a) 所示的结构中，n 个乘法器和一个 n 输入的加法树就能够完成两个 n 维向量的内积操作。基于乘加算子的加速器

表 2.1　现有神经网络加速器的数据流形式

数据流	加速器
基于向量算子的数据流	DianNao [21], DaDianNao [22], PuDianNao [23], Cambricon [24], Cambricon-X [25], Cnvlutin [26], EIE [12], ESE [27]
基于乘加算子的空间数据流	ShiDianNao [28], Eyeriss [29], TPU [30], Neuflow [31] SCNN [32]

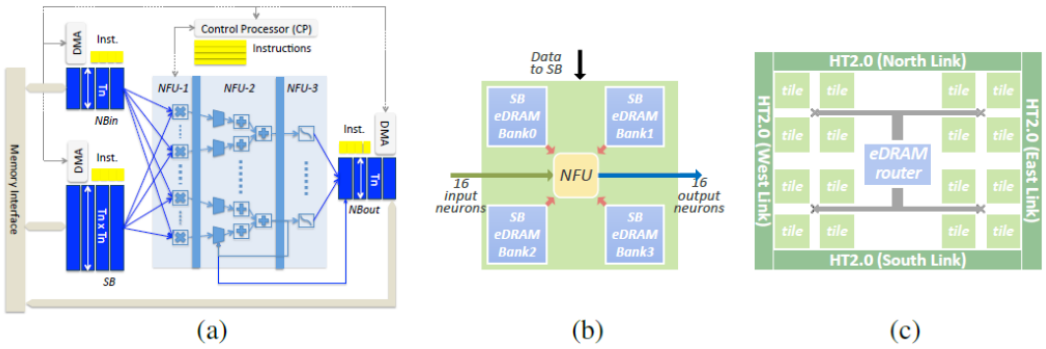

图 2.7　（a）DianNao 加速器结构。（b）DaDianNao 一个 tile 的结构。（c）DaDianNao 一个 node 的结构

（如图 2.6（b））通常具有一系列的规则分布的处理单元，其中每一个处理单元能够完成一次 MAC 操作。处理单元之间采用一种规律的方式进行连接（如二维矩阵），神经元和权值采用某种规律的方式在处理单元之间进行移动，如经典的脉动阵列形式（systolic design），因此这种形式又称为空间数据流形式。表 2.1总结了近年来出现的神经网络加速器和它们的数据流形式。基于向量算子的神经网络加速器包括 DianNao [21],DaDianNao [22],PuDianNao [23],Cambricon [24],Cambricon-X [25],Cnvlutin [26], EIE [12] 和 ESE [27]。基于乘加算子空间数据流的加速器包括 ShiDianNao [28], Eyeriss [29], TPU [30], Neuflow [31] 和 SCNN [32]。

2.3.2　基于向量算子的神经网络处理器

DianNao 是世界上首个深度神经网络处理器，它的结构如图 2.7所示。DianNao 包含以下的主要模块：一个控制器 (Control Processor, 简称 CP)，一个神经功能单元 (Neural Functional Unit, 简称 NFU)，三个片上缓存（NBin，NBout 和 SB）和外部存储接口 (Memory Interface)。CP 采用自定义的指令控制 DianNao 的运行。NBin，NBout 和 SB 分别用来缓存输入神经元，输出神经元和权值。NFU 包含 T_n 个计算单元，每个计算单元包含 T_n 个乘法器和一个 T_n 输入的加法树。在计算过程中每个计算单元共享输入神经元，接收不同的权值，从而计算不同的

表 2.2　现有支持稀疏的神经网络加速器的比较

	权值稀疏	神经元稀疏	注释
Eyeriss [29]	✗	✔	只能跳过计算，无法带来性能提升
Cambircon-X [25]	✔	✗	
Cnvlutin [26]	✗	✔	
EIE [12]	✔	✔	只适合全连接层
ESE [27]	✔	✗	只适合稀疏的 LSTM 层
SCNN [32]	✔	✔	需要额外计算部分和的坐标

输出神经元。因此 NFU 最多读取 T_n 个输入神经元，T_nxT_n 个权值，计算 T_n 个输出神经元。在 TSMC65nm 工艺下，采用 T_n=16 的配置，DianNao 的面积，功耗和吞吐率分别为 $3.02mm^2$，$485mW$ 和 $452GOP/s$，对比与一个 2GHz 的 CPU，能够获得 117 倍的性能提升，并且降低 21 倍的能耗。

DadianNao 是 DianNao 的多核版本，一个 NFU 与外围的存储模块构成一个 tile，多个 tile 通过 fat-tree 进行局部互联组成一个 node，最终多个 node 通过 HyperTransport2.0 互联形成 DaDianNao。DaDianNao 使用 eDRAM 来提供足够的片上缓存用来存储权值和神经元，从而减少片外访存开销，进而有效地处理大规模神经网络。实验显示，在 ST28nm 的工艺下，64 核的 DaDianNao 的面积和功耗分别是 $67.73mm^2$ 和 $15.97W$，对比 Nvidia K20，平均性能提升 21.38 倍，平均能耗降低了 330.56 倍。

PuDianNao 是一个机器学习处理器，它不仅支持神经网络算法，而且支持 K 最近邻算法 (K-Nearest Neighbors，简称 K-NN)，K 均值算法 (K-Means)，线性回归 (Liner Regression, 简称 LR)，支持向量机 (Support Vector Machine，简称 SVM)，朴素贝叶斯 (Naive Bayes，简称 NB) 和决策树 (Classification Tree, 简称 CT) 等多种机器学习算法。PuDianNao 深入分析上述机器学习算法，并且提取出机器学习中的核心运算，其中包括向量内积（LR、SVM 和 DNN), 距离计算（K-NN 和 K-Means），计数（CT 和 NB），排序（K-NN 和 K-Means），非线性函数（如 sigmoid 和 tanh）等。PuDianNao 的核心模块是机器学习单元（Machine Learning Unit，简称 MLU）, 如图 x 所示，MLU 包括六个流水级，分别为 Counter，Adder，Multiplier，Adder Tree，Acc 和 Misc，通过这六个流水级的组合，PuDianNao 能够完成机器学习的核心运算。实验显示，在 65nm 的共一下，PuDianNao 的面积和功耗分别为 $3.51mm^2$ 和 $596mW$ 分为对比 NVIDIA K20 能够提高 1.2 倍的性能并且减少 129.41 倍的能耗。

然而随着神经网络算法的飞速发展，一些新的层类型和新的操作不断涌现，如 ROI Pooling 层，Deconvolution 层等，这又迫使研究者开发新的加速器和指令集来支持这些新的层和操作。为了进一步提高神经网络加速器的通用性，Liu 等人提出了针对神经网络处理器的指令集 Cambricon。Cambricon 的主要思想是为神经网络加速器提供一系列的基本算子（即指令），然后通过这些基本算子逐步搭建完成神经网络的运算。Cambricon 中集成了控制，数据传输，运算和逻辑操作这四种不同的指令，其中运算指令进一步分为矩阵运算，向量运算和标量运算指令，研究者只需要基本指令就能完成神经网络的运算。基于 Cambricon 指令集设计的加速器能够比 DaDianNao 拥有更强的通用性，在 10 个 benchmark 中，DaDianNao 只能支持其中的三种，而基于 Cambricon 的加速器能够全部支持。同时对比与 Intel Xeon E5-2620 和 NVIDIA K40，加速器能够分别获得 91.72 倍和

3.09 倍的性能提升。

2.3.3　基于乘加算子空间数据流的神经网络处理器

ShiDianNao 是面向嵌入式设备的神经网络处理器，实现端到端的神经网络应用。ShiDianNao 的架构如图 x 所示，ShiDianNao 与 DianNao 最大的不同是 NFU 模块。ShiDianNao 的 NFU 是一个大小为 $P_x \times P_y$ 的二维处理单元阵列 (Processing Elements, 简称 PEs)，它采用脉动阵列的形式完成神经网络矩阵向量运算操作。每个 PE 在每个周期能够完成卷积层、全连接层的一次乘法和一次加法，或者平均池化层的一次加法或者最大池化层的比较操作。每个 PE 能够从右方邻居和下方邻居读取神经元，并且能将部分和传播给相邻的 PE，值得注意的是权值通过广播的形式传送给 PE。这种方法能够充分复用神经元和权值，从而提高性能并降低访存能耗。在 $P_x = P_y = 8$ 的配置下使用 65nm 工艺，ShiDianNao 的主频，面积和功耗分别为 $1GHz$,$4.86mm^2$ 和 $320.1mW$，对比 GPU 和 DianNao 能够分别获得 30 倍和 1.87 倍的加速比，减少 4700 倍和 60 倍的能耗。除了 ShiDianNao，Farabet 等人提出的 Neuflow,Chen 等人提出的 Eyeriss 和谷歌的 TPU(tensor processing unit) 均采用二维网格的形式排列处理单元，并且采用脉动阵列的形式完成神经网络运算。其中 Google 的 TPU 主要用来加速其第二代人工智能系统 TensorFolow 的运行，并且相比于 K80 能够有 15 倍的性能提升。

2.3.4　稀疏神经网络处理器

虽然上述加速器能够以低能耗实现高吞吐量，但它们不能利用现代压缩神经网络的稀疏性和不规则性。最近出现了一些能够支持稀疏的神经网络加速器 [12, 25-27, 29, 32]，但它们都有各自的优缺点，如表 2.2所示。

Eyeriss [29] 应用游程压缩方法（run-length-compression,RLC）对稀疏神经元进行编码，具体来说，它在架构中加入 RLC Encoder 对输出神经元进行压缩，同时采用 RLC Decoder 对输入神经元进行解压缩，从而减少访问 DRAM 的数据量，进而减少 DRAM 的带宽需求和访存能耗。同时 Eyeriss 在 PE 单元中加入控制门，当输入神经元为 0 时，PE 单元将被关闭，这样能够跳过不必要的计算，从而减少计算能耗。然而，这两种方法仅仅能够带来能耗的减少，不会带来性能增益。

Cambricon-X [25] 能够充分挖掘稀疏权值带来的收益，包括减少能耗和提升性能。Cambricon-X 使用步长索引的形式压缩静态的权值，从而减少片外和片上访存能耗开销。同时 Cambricon-X 利用索引模块（indexing module，IM）来挖掘稀疏的特性,IM 能够通过稀疏的权值位置信息过滤掉不需要参与计算的神经元。IM 模块的结构如图 x 所示，首先,IM 模块连续地累加权值索引表的值（也就是图

中的 1132），这些累加后的值就是每个连接相对起始点的位置，然后采用 MUX 的逻辑就能选出与非零权值对应的神经元（即 n1,n2,n5,n7）。Cambricon-X 对比不支持稀疏特性的 DianNao 提高了 7.23 倍的性能，并减少了 6.43 倍的能耗，但是 Cambricon-X 并不能挖掘稀疏神经元带来的收益。

Cnvlutin [26] 能够利用动态神经元稀疏筛选出需要进行计算的权值，从而获得 1.37 倍的加速比，但是 Cnvlutin 不能利用权值稀疏特性。ESE [27] 是实现在 FPGA 上的加速器，它面向的是稀疏的 LSTM 模型，并不适用于稀疏的 CNN 模型；由于 LSTM 模型中使用 Tanh 作为激活函数，因此不存在神经元稀疏的特性，所以 ESE 仅仅能够挖掘权值稀疏的特性。

以上工作只能挖掘权值稀疏性或者神经元稀疏性，但不能同时从两者中受益。EIE [12] 采用稀疏矩阵的行压缩形式（compressed sparse row，简称 CSR）存储稀疏的权值，并且通过 LZND（leading non-zero detection）筛选出非零的神经元，使得使得 EIE 能够同时利用神经元稀疏性和权值稀疏性，对比与 DaDianNao 能够提高 2.9 倍的性能，减少 19 倍的能耗并且缩小 3 倍的面积。但是这种架构仅仅针对全连接层的计算，并不能针对卷积层进行计算。

SCNN [32] 能够同时利用神经元稀疏性和权值稀疏性。SCNN 中的计算单元构成了一个二维网格的形式，每个计算单元执行非零神经元和非零权值的乘法操作，同时计算非零乘积的坐标。非零乘积通过坐标在二维计算网络进行路由，最终被分配给对应的累加器阵列，每个非零乘积通过读取修改写入操作对包含部分和的本地 RAM 执行累加，最终获得输出神经元。但是这种架构需要不断计算坐标，大大增加了计算成本和存储成本。因此 SCNN 在处理稠密神经网络时的会损失 21% 的性能，同时增加 33% 的能耗；在处理稀疏神经网络时仅仅能够增加 2.7 倍的性能，减少 2.3 倍的能耗。同时 SCNN 对全连接层和规模为 1x1 的卷积层支持并不理想，在这两种类型的层时时只能利用 20% 的乘法器资源。

2.4　脚注

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ①

① This is a long long long long long long long long long long long long long long long long long long long long long long long long footnote.

第 3 章　稀疏神经网络加速器

3.1　设计原则

在这里，我们将使用一个全连接层作为示例 (参见图 **??**)，分析粗粒度神经网络的特性，进而根据特性分析出神经网络加速器的设计原则。在图中，我们使用 xxx 的剪枝块对规模为 8x4 的全连接层进行剪枝，同时由于 ReLU 的激励原因，神经元 xxxx 的激励为 0。我们采用直接索引的方式存储神经元和权值，即我们仅存储非零元素，并使用比特串来索引非零元素，其中 0 表示对应的位置为零元素，1 表示对应位置为非零元素。通过观察，我们发现了如下的特性。

首先，多个输出神经元之间共享索引信息。如图 **??**，输入神经元之间的连接 $n2, n3, n5, n8$ 和输出神经元 $o1, o2, o3, o4$ 之间的连接被剪除，因此输出神经元共享相同的连接拓扑，也就是说它们共享相同的索引表示 (图 **??**中的 “synapse 索引”)。此外，尽管有单独的突触权重，但所选权重的位置是相同的，因此可以部分共享。

第二，多个输出神经元之间共享输入神经元。如图 **??**，需要参与计算的神经元是 $n1n5$，它们的值被输出神经元共享。不失一般性，在完全连接的层中，设定修剪块大小为 (B_{in}, B_{out})，那么 B_{out} 个相邻输出神经元将共享相同的输入神经元。在卷积层中，设定剪枝块的大小为为 $(B_{fin}, B_{fout}, B_x, B_y)$，那么 B_{fout} 个相邻的输出神经元将共享相同的输入神经元。

第三，挖掘动态稀疏性能够进一步提高运算效率。在图中，通过挖掘权值稀疏，输入神经元 n 被删选出来进行计算，同时由于神经元 $n4, n6$ 是零，在运算过程中，它们对输出神经元没有贡献，最后需要进行计算的输入神经元为。。在图中的实例中，通过挖掘权值稀疏性，共需要进行 12 次乘法，9 次加法完成运算；同时挖掘权值稀疏性和动态神经元的稀疏性，只需要进行 xx 次乘法和 xx 次加法。对比在稠密的情况下 32 次乘法和 28 次加法，分别由 xx 倍和 xx 倍的性能提升。因此，利用动态神经元稀疏性是进一步提高效率的关键 (在上面的示例中，几乎是 xx 改进)。注意，即使考虑到动态神经元稀疏，输出神经元仍然共享索引和所选的输入神经元。

第四，多个输出神经元之间的负载是平衡的，因为它们共享相同的输入神经元。对于图 **??**中的示例，每个输出神经元需要进行 2 次乘法和 1 次加法运算，因此可以避免负载不平衡而造成的性能损失 [27]。

因此，在设计加速器时要考虑以下原则，以最大限度地提高加速器的效率：加速器 (1) 能够利用共享的索引信息和共享的输入神经元信息，简化加速器的设

计;（2）能够利用动态稀疏性进一步提高效率;(4) 利用相邻输出神经元之间的负载均衡。

第 4 章　数　　学

4.1　数学符号

模板定义了一些正体（upright）的数学符号：

符号	命令
常数 e	`\eu`
复数单位 i	`\iu`
微分符号 d	`\diff`
arg max	`\argmax`
arg min	`\argmin`

更多的例子：

$$\mathrm{e}^{\mathrm{i}\pi} + 1 = 0 \tag{4.1}$$

$$\frac{\mathrm{d}^2 u}{\mathrm{d}t^2} = \int f(x)\,\mathrm{d}x \tag{4.2}$$

$$\arg\min_{x} f(x) \tag{4.3}$$

4.2　定理、引理和证明

定义 4.1　If the integral of function f is measurable and non-negative, we define its (extended) **Lebesgue integral** by

$$\int f = \sup_{g} \int g, \tag{4.4}$$

where the supremum is taken over all measurable functions g such that $0 \leqslant g \leqslant f$, and where g is bounded and supported on a set of finite measure.

例 4.1　Simple examples of functions on $\mathbb{R}^d$ that are integrable (or non-integrable) are given by

$$f_a(x) = \begin{cases} |x|^{-a} & \text{if } |x| \leqslant 1, \\ 0 & \text{if } x > 1. \end{cases} \tag{4.5}$$

$$F_a(x) = \frac{1}{1+|x|^a}, \qquad \text{all } x \in \mathbb{R}^d. \tag{4.6}$$

Then f_a is integrable exactly when $a < d$, while F_a is integrable exactly when $a > d$.

引理 4.1 (Fatou)　Suppose $\{f_n\}$ is a sequence of measurable functions with $f_n \geqslant 0$. If $\lim_{n\to\infty} f_n(x) = f(x)$ for a.e. x, then

$$\int f \leqslant \liminf_{n\to\infty} \int f_n. \tag{4.7}$$

注　We do not exclude the cases $\int f = \infty$, or $\liminf_{n\to\infty} f_n = \infty$.

推论 4.2　Suppose f is a non-negative measurable function, and $\{f_n\}$ a sequence of non-negative measurable functions with $f_n(x) \leqslant f(x)$ and $f_n(x) \to f(x)$ for almost every x. Then

$$\lim_{n\to\infty} \int f_n = \int f. \tag{4.8}$$

命题 4.3　Suppose f is integrable on $\mathbb{R}^d$. Then for every $\epsilon > 0$:

i. There exists a set of finite measure B (a ball, for example) such that

$$\int_{B^c} |f| < \epsilon. \tag{4.9}$$

ii. There is a $\delta > 0$ such that

$$\int_E |f| < \epsilon \qquad \text{whenever } m(E) < \delta. \tag{4.10}$$

定理 4.4　Suppose $\{f_n\}$ is a sequence of measurable functions such that $f_n(x) \to f(x)$ a.e. x, as n tends to infinity. If $|f_n(x)| \leqslant g(x)$, where g is integrable, then

$$\int |f_n - f| \to 0 \qquad \text{as } n \to \infty, \tag{4.11}$$

and consequently

$$\int f_n \to \int f \qquad \text{as } n \to \infty. \tag{4.12}$$

证明　Trivial. □

4.3　自定义

Axiom of choice　Suppose E is a set and E_α is a collection of non-empty subsets of E. Then there is a function $\alpha \mapsto x_\alpha$ (a "choice function") such that

$$x_\alpha \in E_\alpha, \qquad \text{for all } \alpha. \tag{4.13}$$

Observation 1　Suppose a partially ordered set P has the property that every chain has an upper bound in P. Then the set P contains at least one maximal element.

A concise proof　Obvious. □

第 5 章　浮　动　体

5.1　三线表

三线表是《撰写手册》推荐使用的方式，如表 5.1。

表 5.1　这里是表的标题

操作系统	TeX 发行版
所有	TeX Live
macOS	MacTeX
Windows	MikTeX

注：一个很长长的表注。

5.2　长表格

超过一页的表格要使用专门的 `longtable` 环境（表 5.2）。

表 5.2　长表格演示

名称	说明	备注
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC

续下页

表 5.2　长表格演示（续）

名称	说明	备注
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC
AAAAAAAAAAAA	BBBBBBBBBBB	CCCCCCCCCCCCCC

5.3　插图

有的同学可能习惯了“下图”、“上表”这样的相对位置引述方式，希望浮动体放在固定位置。事实上，这是不合理的，因为这很容易导致大片的空白。在科技论文中，标准的方式是“图5.1”、“表 5.1”这样的因数方式。

图 5.1　测试图片

关于更多的插图方式，arXiv 上的大部分文献会提供 TEX 源码，大家可以参考学习。

5.4　算法环境

模板中使用 `algorithm2e` 宏包实现算法环境。关于该宏包的具体用法，请阅读宏包的官方文档。

注意，我们可以在论文中插入算法，但是插入大段的代码是愚蠢的。然而这并不妨碍有的同学选择这么做，对于这些同学，建议用 listings 宏包。

Data: this text
Result: how to write algorithm with LaTeX2e
initialization;
while *not at end of this document* **do**
    read current;
    if *understand* **then**
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end

算法 5.1: 算法示例 1

第 6 章　引用文献标注方法

6.1　顺序编码制

6.1.1　角标数字标注法

6.2　其他形式的标注

参 考 文 献

[1] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[2] IOFFE S, SZEGEDY C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[3] BA J L, KIROS J R, HINTON G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450, 2016.

[4] DMITRY U, ANDREA V, VICTOR L. Instance normalization: The missing ingredient for fast stylization[J]. arXiv preprint arXiv:1607.08022, 2016.

[5] WU Y, HE K. Group normalization[J]. arXiv preprint arXiv:1803.08494, 2018.

[6] GOODFELLOW I, BENGIO Y, COURVILLE A, et al. Deep learning: volume 1[M]. MIT press Cambridge, 2016.

[7] KÖSTER U, WEBB T, WANG X, et al. Flexpoint: An adaptive numerical format for efficient training of deep neural networks[C]//Advances in Neural Information Processing Systems. 2017: 1742-1752.

[8] COURBARIAUX M, BENGIO Y, DAVID J P. Binaryconnect: Training deep neural networks with binary weights during propagations[C]//Advances in neural information processing systems. 2015: 3123-3131.

[9] COURBARIAUX M, HUBARA I, SOUDRY D, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1[J]. arXiv preprint arXiv:1602.02830, 2016.

[10] RASTEGARI M, ORDONEZ V, REDMON J, et al. Xnor-net: Imagenet classification using binary convolutional neural networks[C]//European Conference on Computer Vision. Springer, 2016: 525-542.

[11] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network[C]//Advances in Neural Information Processing Systems. 2015: 1135-1143.

[12] HAN S, LIU X, MAO H, et al. Eie: efficient inference engine on compressed deep neural network[C]//Proceedings of the 43rd International Symposium on Computer Architecture. IEEE Press, 2016: 243-254.

[13] WANG Y, XU C, YOU S, et al. Cnnpack: Packing convolutional neural networks in the frequency domain[C]//Advances In Neural Information Processing Systems. 2016: 253-261.

[14] HE T, FAN Y, QIAN Y, et al. Reshaping deep neural network for fast decoding by node-

pruning[C]//Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014: 245-249.

[15] SRINIVAS S, BABU R V. Data-free parameter pruning for deep neural networks[J]. arXiv preprint arXiv:1507.06149, 2015.

[16] HU H, PENG R, TAI Y W, et al. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures[J]. arXiv preprint arXiv:1607.03250, 2016.

[17] MARIET Z, SRA S. Diversity networks[J]. arXiv preprint arXiv:1511.05077, 2015.

[18] DENTON E L, ZAREMBA W, BRUNA J, et al. Exploiting linear structure within convolutional networks for efficient evaluation[C]//Advances in Neural Information Processing Systems. 2014: 1269-1277.

[19] JADERBERG M, VEDALDI A, ZISSERMAN A. Speeding up convolutional neural networks with low rank expansions[J]. arXiv preprint arXiv:1405.3866, 2014.

[20] LEBEDEV V, GANIN Y, RAKHUBA M, et al. Speeding-up convolutional neural networks using fine-tuned cp-decomposition[J]. arXiv preprint arXiv:1412.6553, 2014.

[21] CHEN T, DU Z, SUN N, et al. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[C/OL]//Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS). 2014: 269-284. http://dl.acm.org/citation.cfm?id=2541967.

[22] CHEN Y, LUO T, LIU S, et al. Dadiannao: A machine-learning supercomputer[C]// Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2014: 609-622.

[23] LIU D, CHEN T, LIU S, et al. Pudiannao: A polyvalent machine learning accelerator[C]// ACM SIGARCH Computer Architecture News: volume 43. ACM, 2015: 369-381.

[24] LIU S, DU Z, TAO J, et al. Cambricon: An instruction set architecture for neural networks [C]//ACM SIGARCH Computer Architecture News: volume 44. IEEE Press, 2016: 393-405.

[25] ZHANG S, DU Z, ZHANG L, et al. Cambricon-x: An accelerator for sparse neural networks [C]//Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016: 1-12.

[26] ALBERICIO J, JUDD P, HETHERINGTON T, et al. Cnvlutin: Ineffectual-neuron-free deep neural network computing[C]//Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on. IEEE, 2016: 1-13.

[27] HAN S, KANG J, MAO H, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga[C]//Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2017: 75-84.

[28] DU Z, FASTHUBER R, CHEN T, et al. Shidiannao: Shifting vision processing closer to the

sensor[C]//ACM SIGARCH Computer Architecture News: volume 43. ACM, 2015: 92-104.

[29] CHEN Y H, KRISHNA T, EMER J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE Journal of Solid-State Circuits, 2017, 52(1): 127-138.

[30] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture. ACM, 2017: 1-12.

[31] FARABET C, MARTINI B, CORDA B, et al. Neuflow: A runtime reconfigurable dataflow processor for vision[C]//Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on. IEEE, 2011: 109-116.

[32] ANGSHUMAN P, MINSOO R, ANURAG M, et al. Scnn: An accelerator for compressed-sparse convolutional neural networks[J]. In 44th International Symposium on Computer Architecture, 2017.

附录A 论 文 规 范

致　谢

在研究学习期间，我有幸得到了三位老师的教导，他们是：我的导师，中国科大 XXX 研究员，中科院 X 昆明动物所马老师以及美国犹他大学的 XXX 老师。三位深厚的学术功底，严谨的工作态度和敏锐的科学洞察力使我受益良多。衷心感谢他们多年来给予我的悉心教导和热情帮助。

感谢 XXX 老师在实验方面的指导以及教授的帮助。科大的 XXX 同学和 XXX 同学参与了部分试验工作，在此深表谢意。

在读期间发表的学术论文与取得的研究成果

已发表论文

1. A A A A A A A A A
2. A A A A A A A A A
3. A A A A A A A A A

待发表论文

1. A A A A A A A A A
2. A A A A A A A A A
3. A A A A A A A A A

研究报告

1. A A A A A A A A A
2. A A A A A A A A A
3. A A A A A A A A A