**基于Torch平台的神经网络压缩**

**研究与应用**

**需求文档**

Version 1.5

小组成员：

陈伟民

付强

曹进

李恬霖

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 变更时间 | 修改人 | 审核人 | 备注 |
| 1.0 | 20170324 | 陈伟民 | 付强 曹进 李恬霖 | 初稿 |
| 1.1 | 20170331 | 李恬霖 | 付强 曹进 陈伟民 | 二稿 |
| 1.2 | 20170331 | 曹进 | 付强 李恬霖 陈伟民 | 三稿 |
| 1.3 | 20170331 | 陈伟民 | 付强 李恬霖 曹进 | 四稿 |
| 1.4 | 20170331 | 付强 | 陈伟民 李恬霖 曹进 | 五稿 |
| 1.5 | 20170407 | 陈伟民 | 付强 李恬霖 曹进 | 增加业务需求与扩展需求 |
|  |  |  |  |  |

**目 录**

**[1 前言 1](#_Toc478724072)**

[1.1 目的 1](#_Toc478724073)

[1.2 系统概述 1](#_Toc478724074)

[1.3 文档概述 1](#_Toc478724075)

[1.4 术语和缩略语 2](#_Toc478724076)

**[2引用文档 2](#_Toc478724077)**

**[3需求分析 2](#_Toc478724078)**

[3.1 功能性需求分析 2](#_Toc478724079)

[3.1.1 丰富的工具包 2](#_Toc478724080)

[3.1.2 模块化搭建神经网络 3](#_Toc478724081)

[3.2 非功能性需求分析 4](#_Toc478724082)

[3.2.1 高效性 4](#_Toc478724083)

[3.2.2 用户友好性 4](#_Toc478724084)

[3.2.3 可修改性 4](#_Toc478724085)

[3.2.4 鲁棒性 5](#_Toc478724086)

[3.3 需求识别 5](#_Toc478724087)

[3.3.1 载入数据 5](#_Toc478724088)

[3.3.2 定义神经网络模型 6](#_Toc478724089)

[3.3.3 训练网络 7](#_Toc478724090)

[3.3.4 测试网络 8](#_Toc478724091)

[3.4 RUCM模型 8](#_Toc478724092)

[3.4.1 文件读取数据 9](#_Toc478724093)

[3.4.2 导入工具包 10](#_Toc478724094)

[3.4.3 定义神经网络模型 11](#_Toc478724095)

[3.4.4 训练网络 12](#_Toc478724096)

[3.4.5 随机生成数据 13](#_Toc478724097)

[3.4.6 测试网络 14](#_Toc478724098)

**[4运行要求 14](#_Toc478724099)**

[4.1 硬件要求 14](#_Toc478724100)

[4.2 软件要求 15](#_Toc478724101)

# 前言

## 目的

为了便于协调组内成员进行后期的工作，对项目进行跟踪和监控，对任务的进度进行安排与调控，故对后期工作进行计划。提出需求，指导后续工作。

## 系统概述

Torch 是一个用 Lua 编写的支持机器学习算法的计算框架。其中的一些版本被 Facebook、Twitter 这样的大型科技公司使用，为内部团队专门化其深度学习平台。Lua 是一种在上世纪 90 年代早期在巴西开发出来的多范式的脚本语言。

Torch的目标是在建立科学算法的同时，要有最大的灵活性和速度，而这一过程非常简单。Torch拥有一个大社区驱动包的生态系统，涉及机器学习、计算机视觉、信号处理、并行处理、图像、视频、音频和网络等，并建立在Lua社区基础之上。

Torch的核心是流行的神经网络和优化库，它们易于使用，同时在实现复杂的神经网络拓扑结构时具有最大的灵活性。可以建立任意的神经网络图，并在CPUs和GPUs上有效地并行化。

Torch 7 虽然强大，却并未被基于 Python 的学术社区和通用语言为 Java 的企业软件工程师普遍使用。Deeplearning4j 使用 Java 编写，这反映了我们对产业和易用性的关注。我们相信可用性的限制给深度学习的广泛使用带来了阻碍。我们认为 Hadoop 和 Spark 这样的开源分布式应该自动具备可扩展性。我们相信一个商业化支撑下的开源框架是保证工具有效并建立一个社区的合适解决方案。

## 文档概述

文档用途：本文档主要是介绍Torch系统需求及规格说明。 主要内容：

⮚以用例图、状态图的形式给出 Torch系统功能需求的分解结构，并对用例模型中的参与者和用例进行详细的描述，其中主要包括软件系统的用例模型、系统的核心流程等；

⮚使用 RUCM 模型对功能需求进行建模；

⮚描述了与此次系统实施相关的硬件环境的一些要求；

⮚描述了与此系统实施相关的软件环境的要求；

## 术语和缩略语

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 术语 | 英文 | 说明 |
| 1 | UCM | UCM | 用例建模 |
| 2 | RUCM | RUCM | 限制性用例模型 |
| 3 | LuaJIT | LuaJIT | LuaJIT即采用C语言写的Lua代码的解释器 |
| 4 | Lua | Lua | Lua是一个小巧的脚本语言。由标准C编写而成，几乎在所有操作系统和平台上都可以编译，运行 |
| 5 | nn | nn | Torch中所包含的关于神经网络的一个包，主要提供了搭建神经网络的各种方法 |
| 6 | CUDA | CUDA | CUDA是一种由NVIDIA推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题 |
| 7 | 容器 | Container | Torch中的一种模式，用于向其中添加层来构建复杂的神经网络 |
| 8 | 卷积层 | Convolution layer | 用一个过滤器以步长对图像进行采样并处理 |
| 9 | 全连接层 | Linear layer | n-1层的任意一个节点，都和第n层所有节点有连接。即第n层的每个节点在进行计算的时候，激活函数的输入是n-1层所有节点的加权 |

图表 1‑1 术语和缩略语

# 引用文档

Torch官方关于nn包的文档https://github.com/torch/nn/tree/master/doc

# 需求分析

## 业务需求分析

### 初学者

对于深度学习初学者（例如学生），需要应用抽象程度较高的编程工具，屏蔽底层细节，注重于实现简单的神经网络进行实验。因此不需要十分复杂的平台，只需提供基本的神经网络组件即可。

### 科研人员

作为一个深度学习工作者，通常需要即时、简便地构造一个神经网络（卷积神经网络、循环神经网络甚至更复杂的网络结构），另外用包装好的工具包（常见的损失函数和优化算法）对神经网络进行训练，并对网络进行测试，当网络结构过于庞大，需要迁移至GPU训练时，迁移过程需要尽可能简单，甚至不需要了解基于CUDA编程的细节与方法。

另外，工具包应当具有灵活性，源代码满足易读、易更改、易扩展等要求。开源平台可以实现使用者即时的扩展功能，并按自己的需求对模块进行修改。

### 深度学习领域

随着神经网络相关研究的发展，深度网络等复杂的网络结构对于解决某些实际问题往往有着更优秀的表现。搭建深度网络是一个复杂又耗时的工程，同时，巨大的网络结构造成在训练过程中需要调整大量的参数，这对硬件的存储和计算能力都提出了很高的要求。如果这些问题得以解决，势必将大大推进深度学习领域的研究。甚至在解决存储和计算问题后，深度网络可以迁移至移动端进行训练。

## 功能性需求分析

Torch是一个广泛支持机器学习算法的科学计算框架。易于使用且高效，主要得益于一个简单的和快速的脚本语言Lua，和底层的C / CUDA实现。

### 丰富的工具包

Torch7提供了拥有十分详尽接口的库，Torch有一个在机器学习领域大型生态社区驱动库包，包括计算机视觉软件包，信号处理，并行处理，图像，视频，音频和网络等，基于Lua社区建立。

Torch可以看做是由Lua编写的一个package，本身的后台都是基于C和CUDA写的，并提供Lua接口。一般基于torch的简单开发，可以在Lua的级别下完成。如果考虑到功能的复杂性或者package的效率问题，则需要用C和CUDA来编写后台。

神经网络是Torch中的一个包，如(nn/cunn)，其核心是各个层（torch中称为Module），主要的代码包括Lua, C , CUDA三部分。若一个层所需运算可直接通过Tensor操作来完成，则仅一个lua文件即可，如nn.Linear; 若需要C/CUDA实现，则还需要写C/CUDA代码，如nn.SpatialConvolutionMM。理论上可以用nn里的模块实现任何DAG构造的网络，当然也包括RNN、LSTM之类的。

### 模块化搭建神经网络

图计算框架都有一个限制，就是需要用户把所有的计算全部都表示成一张图来高效运行。这么做不论是在逻辑上，调试上以及和运行环境的交互(python)上面都是有一定劣势。

而Torch采取了支持用户把计算拆分成多步来做，用户可以直接利用Lua来选择下一步执行什么。用户可以比较简单地对计算进行模块分割，并且根据比如输入长度的不同来直接动态改变需要运行哪一个步骤。Torch为代表的过程式计算更加灵活。

Torch7和nn类库拥有清晰的设计和模块化的接口。Torch对卷积网络的支持非常好，Torch通过时域卷积的本地接口使得它的使用非常直观。Torch通过很多非官方的扩展支持大量的递归神经网络，Torch本质上是以图层的方式定义网络的，在Torch中定义新图层非常容易。

作为一个深度学习的框架，Torch允许自由地实现对已有模块逻辑复杂的调用，允许依据需要使用已有的模型（模型的参数自己训练得到），也可以在已有模型的基础上增加自己的layer，或者在顶端选择自己需要的分类器，Torch构建的是一个生态系统，安装新的模型实现模块只需要luarocks install package。

在需要编写系统没有提供的层时，如果所有运算均能通过Tensor自带的操作来完成，这样只需要写一个lua文件，通过require来使用，而使用Tensor操作无法完成或效率太低时，就需要使用C和CUDA来实现核心算法，通过Lua来调用。

## 非功能性需求分析

### 高效性

（1）轻量级系统

Torch平台与该具有与其他开源深度学习框架相比更加轻量级的系统架构，以保证高效运行，未来的发展目标是希望torch 7可以直接部署到手机上。

（2）高效底层语言支持

Torch应该使用效率高且平台兼容性好的语言编写底层代码，例如C或C++。这样不仅可以进一步实现核心计算单元的优化，也保证了Torch做高性能计算、异构计算、以及应用于嵌入式平台的可能性。

（3）支持并发运行

Torch应该支持多线程、多GPU运行，以保证大规模深度神经网络的高效训练和使用。

### 用户友好性

为了保证深度学习研究者和开发者可以尽快熟悉、使用Torch平台，平台应该具有更简洁易懂的操作界面，更简单易学的编程语法，在各个方面更加注重用户友好性。

### 可修改性

Torch作为一个内置丰富软件工具包的开发平台，为深度学习研究者和开发者提供了基础的深度学习模型框架和方法接口。但是由于其提供的框架和方法过于基础，故需要其具有可修改性，以便通过不断地研究，引入新的算法或技术，以实现平台搭建的深度学习网络或实现的其他功能具有更好地时间和空间效率。

### 鲁棒性

系统应该在异常和危险情况下都能保持健壮的表现和稳定的性能，有健全的容错机制和方法。

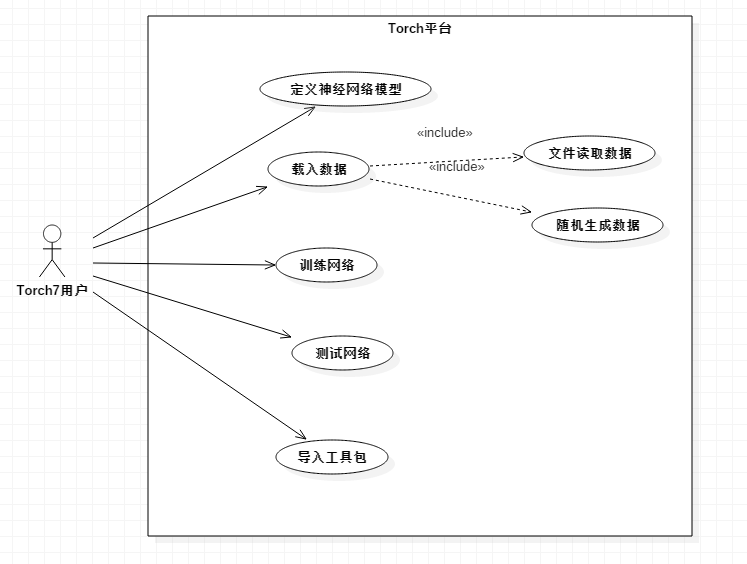
## 扩展需求

神经网络功能强大。但是，其巨大的存储和计算代价也使得其实用性特别是在移动设备上的应用受到了很大限制。降低大型神经网络其存储和计算消耗，使得其甚至可以在移动设备上得以运行，即要实现“深度压缩”。

利用Torch平台提供的模块化实现，我们可以扩展自己的神经网络层（全连接层、卷积层），修改内部的权重计算方式，去掉不必要的连接，进而减少参数的数量或存储，缩短网络训练时间和存储消耗，但仍保证网络训练产生模型的准确性。

## 需求识别

将业务需求分解为功能性需求，同时考虑到非功能性需求，最终得到的Torch的用例图如下图所示。



图表 3‑1 Torch的用例图

### 载入数据

载入数据主要包括两个步骤：

1.首先用户需要引入所需要的包，如cephes、randomkit、nn、cutorch等。Torch是一个工具包的集合，在使用工具前首先要引入工具所在的包，例如torch包提供了最基本的矩阵或表的操作，nn包提供了神经网络组件的实现，cutorch包提供了程序在GPU上运行的基础实现。引入包的操作由require [package]实现；

2.根据不同的需求载入数据，例如可从文件载入，也可使用torch自带的随机方法生成数据。

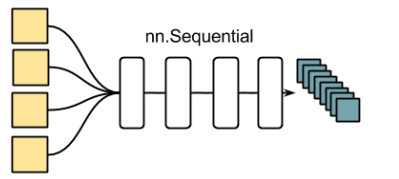
### 定义神经网络模型

生成模型的过程主要分为以下几个步骤（以建立卷积神经网络模型为例）：

1.引入相关的包。

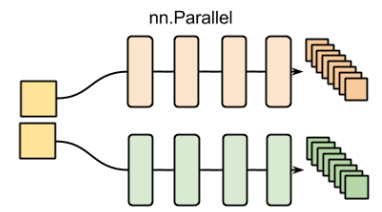
2.选择容器存放模型，如sequential、concat、parallel、bottle等。容器是建立模型的基础，以神经网络为例，首先要定义神经网络的架构，torch提供了三种基本类型的容器：

Sequential：所有层组成一个序列化的神经网络，第一层连接所有输入。由[module]=nn.Sequential()实现。



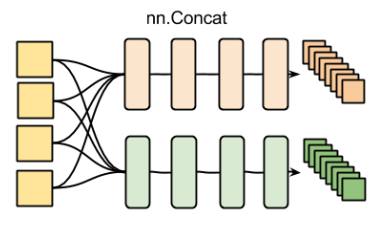
图表 3‑2 Sequential容器

Parallel：针对每一个输入，可以构建一个序列化的神经网络，最后连接输出的结果。由[module]=nn.Parallel(inputDimension, outputDimension)实现。



图表 3‑3 Parallel容器

Concat：构建若干个序列化的神经网络，每个网络的第一层连接所有输入。由[module]=nn.Concat(dim)实现。



图表 3‑4 Concat容器

3.添加层。建立好容器后，就可以向容器中添加不同的层，以构建一个卷即神经网络为例，nn包提供了两个基本的层：

SpatialConvolution：一个二维的卷积层，可对输入数据进行卷积处理。由[module]=nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])实现。其中nInputPlane是输入图像的通道数，nOutputPlane表示卷积层的输出通道数，kW和kH表示卷积核的大小，dW和dH表示卷积核的移动步长，padW和padH表示对输入通道补0的情况。

Linear：线性全连接层，可对输入数据进行线性处理，如y=Ax+b。由[module]=nn.Linear(inputDimension, outputDimension, [bias=true])实现。其中inputDimension和outputDimension分别表示全连接层输入和输出的维度，偏置项默认存在。在向容器中添加层时，可通过[Container]:add(nn.[layer]([params])实现。

4.添加层后，还需要定义隐藏单元的激活函数，如tanh、logsoftmax等。

### 训练网络

训练一个神经网络，包括以下步骤：

1.定义损失函数，例如，MSECriterion用于计算回归问题的损失，ClassNLLCriterion用于计算分类问题的损失。通过[criterion] = nn.MSECriterion()实现。

2.选择训练方法，可以通过自行编写更新权重的算法，也可以调用torch提供的优化方法，例如，随机梯度下降算法等，通过[trainer]=StochasticGradient(module, criterion)实现。

3.训练神经网络，利用定义的损失函数和优化算法训练神经网络，通过trainer:train(dataset)实现。

### 测试网络

测试一个神经网络，可以通过output = [model]:forward(testData)来实现，其中testData是测试数据集。

## RUCM模型

RUCM 即限制性用例建模。 它的目标是：

1．使 UCMs 更加可理解并且更精确。

2．从 UCMs 自动生成分析模型。

RUCM 有以下两部分组成：

1．一个用于系统组织 UCSs 的用例模板。

2．限制用户写 UCSs 的一系列规则。

通过 RUCM 模型能够对用例进行规范的描述，接下来将使用 RUCM 模型描述图1中的用例。

### 文件读取数据





图表 3‑5 文件读取数据RUCM

### 导入工具包

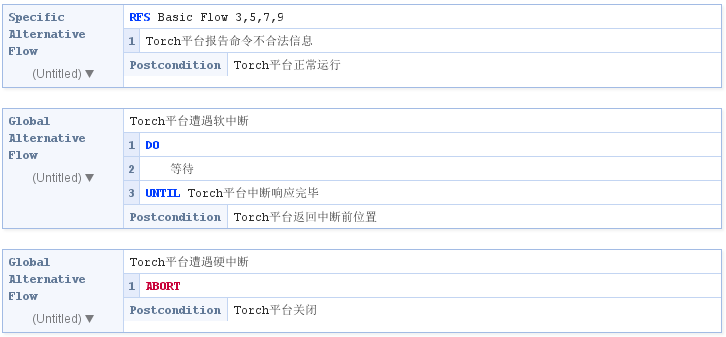




图表 3‑6 导入工具包RUCM

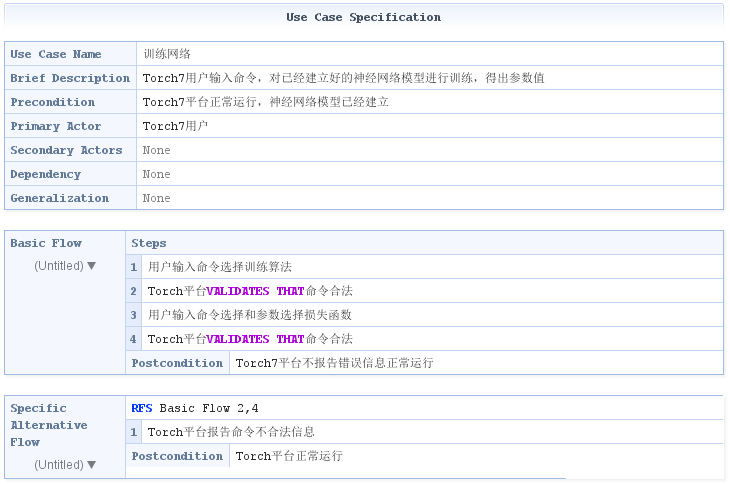
### 定义神经网络模型





图表 3‑7 定义神经网络模型RUCM

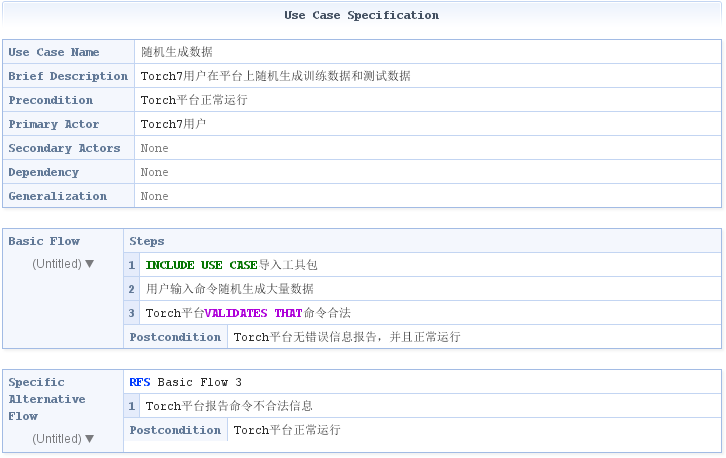
### 训练网络





图表 3‑8 训练网络RUCM

### 随机生成数据





图表 3‑10 随机生成数据RUCM

### 测试网络





图表 3‑11 测试网络RUCM

# 运行要求

## 硬件要求

* CPU: Intel等主流CPU
* 内存：2G内存及以上
* 硬盘：20G硬盘及以上
* GPU：能够使用cuda（即nvidia显卡）

## 软件要求

* 操作系统：linux（包括ubuntu和OS X等）
* 编译环境：luaJit（安装时包括）