目录

[引言 1](#_Toc76394755)

[编写目的 1](#_Toc76394756)

[背景 1](#_Toc76394757)

[定义 1](#_Toc76394758)

[参考资料 2](#_Toc76394759)

[可行性研究的前提 3](#_Toc76394760)

[要求 3](#_Toc76394761)

[目标 3](#_Toc76394762)

[条件，假定和限制 3](#_Toc76394763)

[进行可行性研究的方法 4](#_Toc76394764)

[评价尺度 4](#_Toc76394765)

[对现有系统的分析 5](#_Toc76394766)

[处理流程和数据流程 5](#_Toc76394767)

[局限性 6](#_Toc76394768)

[受建议的系统 7](#_Toc76394769)

[对所建议系统的说明 7](#_Toc76394770)

[处理流程和数据流程 7](#_Toc76394771)

[改进之处 9](#_Toc76394772)

[影响 9](#_Toc76394773)

[社会因素方面的可行性分析 9](#_Toc76394774)

[法律方面的可行性 9](#_Toc76394775)

# 引言

## 编写目的

可行性研究的目的其实是进行一次大大压缩简化了的系统分析和设计过程，在可行性研究的过程中，力争能达到解决一下问题：

1. 使用现有的技术是否能实现这个系统
2. 这个系统的经济效益是否能超过它的开发成本
3. 系统的操作方式是否能在这个用户组织内行得通

再者，可行性分析研究的最根本的目的是对以后的行动方针提出建议，以避免时间，资源，人力和金钱的浪费。

预期读者：

对于此可行性研究报告的读者可以又一下两种：

1. 神经网络模型的使用者，也就是向软件开发组提出需求的科研人员
2. 软件项目组组长，对方将会以技术的角度去分析可行性

## 背景

项目名称：ViTorch（基于pytorch的可视化神经网络编辑器）

本项目的任务提出者：ViTorch项目组

本项目的任务开发者：ViTorch项目组

本项目的用户：

* 短期内，针对那些想使用却不会编写代码编写神经网络，并且功能需求低的用户群体，如：在校大学生和跨领域研究者；
* 长期，针对在日常工作研究中要经常使用神经网络的用户群体，提升他们的工作开发效率，这类人有科研工作者和算法开发人员。

项目目标：建立一个高效、准确，操作方便，具有可视化，本地及时运行神经网络编辑软件，以满足编辑人员进行综合的、形象的、及时的实现、管理神经网络模型。

## 定义

本文档中可能出现的专业术语解析如下：

**PyTorch：**

PyTorch是一个开源的Python机器学习库，基于Torch，用于自然语言处理等应用程序。它是一个基于Python的可续计算包，提供两个高级功能：1、具有强大的GPU加速的张量计算（如NumPy）。2、包含自动求导系统的深度神经网络。

PyTorch的前身是Torch，其底层和Torch框架一样，但是使用Python重新写了很多内容，不仅更加灵活，支持动态图，而且提供了Python接口。它是由Torch7团队开发，是一个以Python优先的深度学习框架，不仅能够实现强大的GPU加速，同时还支持动态神经网络。

PyTorch既可以看作加入了GPU支持的numpy，同时也可以看成一个拥有自动求导功能的强大的深度神经网络。除了Facebook外，它已经被Twitter、CMU和Salesforce等机构采用。

**PyQt：**

PyQt是一个创建GUI应用程序的工具包。它是Python编程语言和Qt库的成功融合。Qt库是目前最强大的库之一。PyQt是由Phil Thompson 开发。

PyQt实现了一个Python模块集。它有超过300类，将近6000个函数和方法。它是一个多平台的工具包，可以运行在所有主要操作系统上，包括UNIX，Windows和Mac。 PyQt采用双许可证，开发人员可以选择GPL和商业许可。在此之前，GPL的版本只能用在Unix上，从PyQt的版本4开始，GPL许可证可用于所有支持的平台。

## 参考资料

<https://pytorch.org/tutorials/>

<https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html>

# 可行性研究的前提

## 要求

功能：

编辑部分：

* 能实现可视化的编辑神经网络，
* 神经网络层之间先后由连线相连，
* 能增加神经网络层，
* 能设置与更改网络层参数，
* 能插入和从中间删除神经网络层，
* 能将编辑好的神经网络图形生成为神经网络对象并生成构建改神经网络的代码，
* 能使用常见神经网络层。

数据集部分：

* 能下载并使用网络上PyTorch自带的数据集，
* 能使用自己制作的数据集，
* 能现场采集数据集，
* 能实现一些提供的数据集的预览效果。

训练部分：

* 能使用挑选的数据集对制作的神经网络进行训练，
* 能设置训练的各个参数，如：正则化，学习率等等，
* 能实时展示训练曲线（包含训练集上误差，和验证集上的误差），
* 能暂停和继续训练。

性能：与PyTorch一致

完成期限：7月8号之前

## 目标

短期目标：

* 能实现简单的视觉神经网络的编写与训练，
* 同时图形界面直观简洁，容易上手，让深度学习的初学者也能快速的完成神经网络模型的编辑。

长期目标：

* 能实现所有现有的神经网络模型的编写，
* 高级用户可以自定义为编辑神经网络层创造全新的网络，
* 可以自定义神经网络模型的损失函数，
* 可以加载PyTorch的神经网络对象将其图形化之后进行再编辑。

## 条件，假定和限制

开发条件：

|  |  |
| --- | --- |
| 条件 | 工具 |
| 开发语言 | Python |
| 神经网络框架 | PyTorch |
| 界面实现 | Pyqt5 |
| 数据库技术 | SQLserver2019 |

环境要求：

|  |  |
| --- | --- |
| 要求项 | 要求 |
| 操作系统 | Windows10及以上 |

经费：免费的开源项目，无收入来源，也没经费预算。

## 评价尺度

长短期的目标完成度，完成时间，界面流畅程度。

# 对现有系统的分析

## 处理流程和数据流程

处理流程：完全的代码编写，

1. 导入头文件：

*import* torch

*import* torch.nn *as* nn

*import* torch.optim *as* optim

*import* torch.nn.functional *as* F

*import* torchvision.datasets *as* dsets

*import* torchvision.transforms *as* transforms

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*import* torch.utils.data *as* data

1. 导入数据集：

batch\_size=64

train\_dataset=dsets.MNIST(*root*='./data',*train*=True,*transform*=transforms.ToTensor(),*download*=True)

test\_dataset=dsets.MNIST(*root*='./data',*train*=False,*transform*=transforms.ToTensor())

train\_loader=data.DataLoader(*dataset*=train\_dataset,*batch\_size*=batch\_size,*shuffle*=True)

test\_loader=data.DataLoader(*dataset*=test\_dataset,*batch\_size*=batch\_size)

1. 编辑网络：

class ConvNet(nn.Module):

    def \_\_init\_\_(*self*):

        super(ConvNet, *self*).\_\_init\_\_()

*self*.conv1 = nn.Conv2d(1, 4, 5, *padding*=2)

*self*.pool = nn.MaxPool2d(2, 2)

*self*.conv2 = nn.Conv2d(depth[0], depth[1], 5, *padding*=2)

*self*.fc1 = nn.Linear(image\_size // 4 \* image\_size // 4 \* depth[1], 512)

*self*.fc2 = nn.Linear(512, num\_classes)

    def forward(*self*, *x*):

*x* = F.relu(*self*.conv1(*x*))

*x* = *self*.pool(*x*)

*x* = F.relu(*self*.conv2(*x*))

*x* = *self*.pool(*x*)

*x* = *x*.view(-1, image\_size // 4 \* image\_size // 4 \* depth[1])

*x* = F.relu(*self*.fc1(*x*))

*x* = F.dropout(*x*, *training*=*self*.training)

*x* = *self*.fc2(*x*)

*x* = F.log\_softmax(*x*, *dim*=0)

*return* *x*

1. 训练神经网络

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), *lr*=0.001, *momentum*=0.9)

record = []

weights = []

*for* epoch *in* range(num\_epoch):

    train\_rights = []

*for* batch\_idx, (data, target) *in* enumerate(train\_loader):

        data, target = data.clone().requires\_grad\_(

            True), target.clone().detach()

        net.train()

        output = net(data)

        loss = criterion(output, target)

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        right = rightness(output, target)

    train\_rights.append(right)

## 局限性

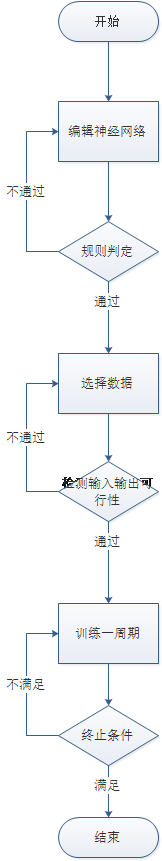
纯代码编辑，首先要对深度学习和神经网络有深度的了解。并且要有一个程度的代码功底，并且训练过程可视化程度低，虽然可以自己编写可视化的代码，但是这无疑会增加代码量，还有，也没法清楚的展示神经网络的结构。

# 受建议的系统

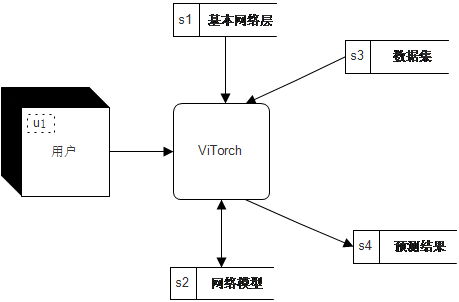
## 对所建议系统的说明

我们使用PyQt搭建可视化界面，以PyTorch为内核，实现界面组件到PyTorch对象的映射，保障了我们的系统的运行效率不输于PyTorch本身的效率。

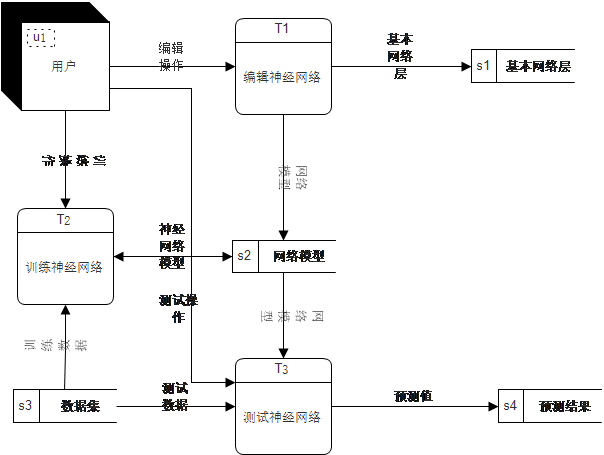
## 处理流程和数据流程



首先，用户开始编辑属于自己的神经网络，接着对神经网络进行测试，若出现异常则更改神经网络直至通过，接着挑选数据集，同时检验数据集的输入输出维度是否和神经网络符合，如果不符合的话，重新选择数据集或者重新编辑神经网络，最后进行不断的训练，直神经网络的精度达到一定的标准，或者到训练速度放缓的基本为0时结束。



我们的ViTorch系统，使用一些基本的网络层来搭建网络模型，并可以在挑选的数据集上进行训练和预测。



## 改进之处

图形化的模型编辑

可视化的训练过程

## 影响

让代码基础不佳的使用者也能轻松的使用神经网络模型解决自己的问题。

# 社会因素方面的可行性分析

## 法律方面的可行性

首先我们的技术都是基于一些开源框架，如：PyQt5和PyTorch。同时，我们的软件也将进行开源，不作为商业用途。