授课教师：胡鹤

报告人：赵元培（组长）

学号：2015202006

小组成员：胡景文 杨文清

人工智能课程总结

yuanpei_z@163.com

目录

项目报告

基于 Arduino 及深度学习算法的智能识路小车设计及实现

摘要

本文简要介绍了基于 Arduino 及深度学习算法的智能识路小车设计及实现。其中包括对小车的执行组件、搭建结构、传感器、Arduino 单片机软件编程及试验结果的介绍。并分为三个阶段实现，在每个阶段均有项目成员分工的具体介绍。

本项目以 Arduino 单片机为控制核心。一方面，利用超声波传感器，检测小车前方障碍物的距离，然后把数据传送给单片机。通过超声波不断的循环检测周边环境的情况进行自动避障；另一方面，使用深度学习算法，建立神经网络模型，对于小车上的摄像头捕捉的图像进行处理，向小车传回指令，实现智能识路。

在软件方面，利用 Arduino 语言进行编程，通过软件编程来控制小车运转。该系统在驱动方面采用 L298N 驱动 2 个直流电机带动小车运行。并且，用 PWM 系统调速,控制小车前进的速度。实现小车根据外部环境,做出前进、后退和转向等动作,从而完成避障的功能，本设计具有有一定的实用价值。

关键词：Arduino 单片机；超声波传感器；深度学习；智能识路

1. 背景介绍

1.1. Arduino 单片机介绍

Arduino，是一个基于开放原始码的软硬件平台，构建于开放原始码 simple I/O 介面版，并且具有使用类似 Java，C 语言的 Processing/Wiring 开发环境。它包含两个主要的部分：硬件部分是可以用来做电路连接和 Arduino 电路板；另外一个则是 Arduino IDE，你的计算机中的程序开发环境。你只要在 IDE 中编写程序代码，将程序上传到 Arduino 电路板后，程序便会告诉 Arduino 电路板要做些什么了。

Arduino 能通过各种各样的传感器来感知环境，通过控制灯光、马达和其他的装置来反馈、影响环境。板子上的微控制器可以通过 Arduino 的编程语言来编写程序，编译成二进制文

件，收录进微控制器。对 Arduino 的编程是利用 Arduino 编程语言（基于 Wiring)和 Arduino 开发环境(based on Processing)来实现的。基于 Arduino 的项目，可以只包含 Arduino，也可以包含 Arduino 和其他一些在 PC 上运行的软件，他们之间进行通信（比如 Flash, Processing, MaxMSP)来实现。可以自己动手制作，也可以购买成品套装;Arduino 所使用到的软件都可以免费下载. 硬件参考设计（CAD 文件)也是遵循 availableopen-source 协议，你可以非常自由地根据你自己的要求去修改他们.

1.2. 项目主要内容

本设计题目为《基于 Arduino 及深度学习算法的智能识路小车设计及实现》，以 Arduino 单片机为控制核心。一方面，利用超声波传感器，检测小车前方障碍物的距离，然后把数据传送给单片机，人工设计算法判断不同情况下小车的移动情况，通过超声波不断的循环检测周边环境的情况进行自动避障；另一方面，使用深度学习算法，建立神经网络模型，对于小车上的摄像头捕捉的图像进行处理，向小车传回指令，实现智能识路。

2. 总体设计

2.1. 设计原理与方法

基本移动：

小车通过两个马达控制车轮转动，可以控制车轮速度和方向，其中速度可以为负，表示倒车

简单避障：

小车通过头部加装的超声波传感器进行距离探测，再通过单片机烧录的程序产生对应动作

视频传输：

小车通过头部加装的摄像头进行视频采集，通过特定软件实时传递至电脑端，实现视频传输

智能识路：

通过基于 Sikit-Learn 以及 TensorFlow 建立的深度学习模型对视频中关键帧进行分类，判断小车下一步的状态，将结果通过蓝牙传输至单片机上，从而实现智能识路

2.2. 硬件设计

本小车的硬件部分主要分为几个模块：蓝牙传感器、超声波传感器、Arduino 单片机、电源、两个直流电动机、电机驱动板、车身。电源连接在 Arduino 单片机上给整个小车供电。小车以 Arduino 单片机为核心，连接电机驱动板控制两个直流电动机的运转，从而实现小车的移动。将超声波传感器安置在车身的最前端，用于探测前方是否有障碍物。当超声波传感器遇到障碍物，将反馈提供到单片机里从而做出相应反应，从而实现整个小车的避障功能。蓝牙传感器实现小车和电脑信号的连接，从而为基于深度学习的智能避障功能打下基础。

2.3. 软件设计

2.3.1. Arduino 语言

Arduino 是一款便捷灵活、方便上手的开源电子原型平台。包含硬件（各种型号的 Arduino 板）和软件（Arduino IDE)。由一个欧洲开发团队于 2005 年冬季开发。它构建于开放原始码 simple I/O 介面版，并且具有使用类似 Java、C 语言的 Processing/Wiring 开发环境。主要包含两个主要的部分：硬件部分是可以用来做电路连接的 Arduino 电路板；另外一个则是 Arduino IDE，你的计算机中的程序开发环境。你只要在 IDE 中编写程序代码，将程序上传到 Arduino 电路板后，程序便会告诉 Arduino 电路板要做些什么了。

Arduino 能通过各种各样的传感器来感知环境，通过控制灯光、马达和其他的装置来反馈、影响环境。板子上的微控制器可以通过 Arduino 的编程语言来编写程序，编译成二进制文件，烧录进微控制器。对 Arduino 的编程是通过 Arduino 编程语言 (基于 Wiring)和 Arduino 开发环境(基于 Processing)来实现的。基于 Arduino 的项目，可以只包含 Arduino，也可以包含 Arduino 和其他一些在 PC 上运行的软件，他们之间进行通信 (比如 Flash, Processing, MaxMSP)来实现。

2.3.2. TensorFlow

TensorFlow 是一个采用数据流图（data flow graphs），用于数值计算的开源软件库。节点（Nodes）在图中表示数学操作，图中的线（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。它灵活的架构让你可以在多种平台上展开计算，例如台式计算机中的一个或多个 CPU（或 GPU），服务器，移动设备等等。TensorFlow 最初由 Google 大脑小组（隶属于 Google 机器智能研究机构）的研究员和工程师们开发出来，用于机器学习和深

度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。

2.4. 前期准备

主体清单

序号	名称	规格	位号	用量		序号	名称	规格	位号	用量
1	控制板	Arduino UNO R3		1		9	电机驱动模块	298		1
2	转接板	UNO 功能转接板	ZYV6 转接板	1		10	3MM 底盘	透明 2 驱		1
5	杜邦线	4P 母对母 20CM	4P 整条	1		11	电池盒转接板	亚克力材质		1
6	杜邦线	2P 公对母 20CM	单根的	1		12	USB 方头线			1
7	65MM 橡胶轮子			2		13	电机 1:48	双轴直流减速电机		2
8	电机固定片	亚克力电机固定片		4		14	6 节电池盒			1

额外附件清单

序号	名称	规格	备注	用量		序号	名称	规格	备注	用量
1	M3*25 通孔铜柱			8		3	清单一张			1
2	M3*10 通孔铜柱			4		3	2P 20CM 线		红色剪掉端子	4

2.5. 任务分解

阶段	任务
第一阶段：随机行走	硬件组装
	熟悉 IDE
	烧录程序，控制避障
第二阶段：传感避障	蓝牙串口的安装
	实现手机端利用蓝牙发送讯号控制小车
	完成 PC 端视频讯号的传递（DroidCamApp）

	学习利用 python 处理图片、视频
	实现并对比一些简单的计算图片相似度算法，为下一阶段提供思路
第三阶段：深度学习识路	定义寻路具体问题
	收集训练、测试集
	用视频图片处理算法处理训练集
	利用测试集测试不同深度学习算法
	选定算法，利用蓝牙传回小车
	修改 arduino 程序，直到实际测试满意

3. 第一阶段：随机行走

3.1. 阶段目标

- 完成硬件的组装搭建
- 熟悉软件（IDE 等）的安装使用
- 编写程序，控制小车前进，遇到障碍物能够回避

3.2. 结构说明

- **小车底板：**左右 2 个轮子，分别由一个 motor 控制
- **Arduino uno 板：**用于烧录程序，连接驱动板等设备
- **L293D 驱动板：**连接 2 个 motor，连接蓝牙串口，连接超声波传感器
- **HC-SR04 超声波传感器：**探测前方障碍物，将数据传回主板
- **BT06 蓝牙串口：**提供远程控制接口（暂未使用）
- **电池：**连接驱动板，提供电源
- **电动机** 2 个
- **面包板** 1 个
- **连接线**若干

以下是组装过程：

1. 将【Arduino UNO 主板】与【L293D 电机驱动扩展板】连接
2. 将【小车底座、车轮】组装好，与【Arduino UNO 主板】连接
3. 将【电动机 2 个】与【小车底座】组装好，与【L293D 电机驱动扩展板】连接
4. 利用连接线，将【蓝牙接收器】接入【L293D 电机驱动扩展板】的舵机接口
5. 利用连接线，将【超声波距离探测器】接入【L293D 电机驱动扩展板】的舵机接口
6. 利用连接线，通过【面包板】，将【电源】接入【L293D 电机驱动扩展板】
7. 固定以及焊接各个部分

3.3. 接口说明

- **电动机接口：**
  两个电动机分别接入【L293D 电机驱动扩展板】的【M1】【M4】接口

- **蓝牙接口：**
  【蓝牙接收器】接入舵机接口，编号为【9】

- **距离探测器接口：**
  【超声波距离探测器】的【trig】 端接入舵机【9】接口
  【超声波距离探测器】的【echo】端接入舵机【10】接口

3.4. 程序说明

- **源代码：**

```
#include "AFMotor.h"

//设置两个 motor 的接口和频率
AF_DCMotor motor_left(1,MOTOR12_1KHZ);
AF_DCMotor motor_right(4,MOTOR34_1KHZ);

//设置两个舵机接口
const int TrigPin = 9;
const int EchoPin = 10;

//距离变量
float cm;

void setup()
{
  Serial.begin(9600);
  pinMode(TrigPin, OUTPUT);
  pinMode(EchoPin, INPUT);
  //设置左右 motor 速度
  motor_left.setSpeed(180);
  motor_right.setSpeed(210);
}

void loop()
{
  digitalWrite(TrigPin, LOW); //低高低电平发一个短时间脉冲去 TrigPin
  delayMicroseconds(2);
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);

  cm = pulseIn(EchoPin, HIGH) / 58.0; //将回波时间换算成 cm
  cm = (int(cm * 100.0)) / 100.0; //保留两位小数

  if(cm < 15){
    motor_left.setSpeed(180);
    motor_left.run(FORWARD);
    motor_right.run(FORWARD);
    delay(300);
  }
  else if(cm < 40){
    motor_left.setSpeed(0);
    motor_left.run(BACKWARD);
    motor_right.run(BACKWARD);
    delay(300);
  }
  else{
    motor_left.setSpeed(180);
    motor_left.run(BACKWARD);
    motor_right.run(BACKWARD);
  }
  delay(100);
}
```

- **代码说明：**
  1. 正常运行：
     设置左轮速度 180，右轮速度 210（由于 M1 和 M4 频率不同，需要微调）

  2. 面对正常障碍物（15~40cm）：
     设置左轮速度 0，实现左转

  3. 障碍物过近（0~15cm）
     反向旋转，实现倒车

3.5. 实验结果

- 基本能够沿直线运动
- 遇到远处障碍物能够转向
- 无法转向时能够倒车

3.6. 人员分工

- 赵元培：熟悉 IDE，编写、调试程序，展示阶段成果
- 胡景文：硬件组装，调试程序
- 杨文清：硬件组装

3.7. 存在的不足

- 两个 motor 存在细微误差，不能完全按照直线运动
- 超声波传感器位置较高，遇到较低障碍物无法检测并避开
- 超声波传感器宽度不足，小车两侧边缘可能被卡住

4. 第二阶段：传感避障

4.1. 阶段目标

- 在第一阶段的基础上，完善小车蓝牙串口的安装
- 实现手机端利用蓝牙发送讯号控制小车
- 完成手机与 PC 端视频讯号的传递（DroidCamApp）
- 学习利用 python 处理图片、视频
- 实现并对比一些简单的计算图片相似度算法，为下一阶段提供思路

4.2. 蓝牙串口说明

1. BT06 串口模块

   - VCC：接 Arduino 的 5V
   - GND：接 Arduino 的 GND
   - TXD：一般表示为自己的发送端，接 Arduino 的 RX
   - RXD：一般表示为自己的接收端，接 Arduino 的 TX

2. 手机端控制

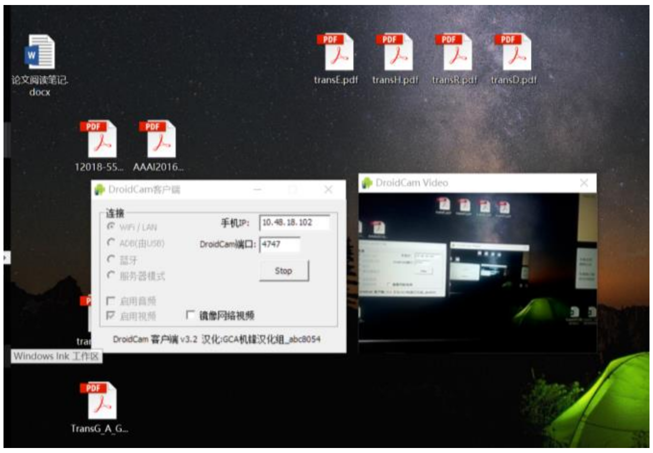
   - 使用软件：蓝牙小车 APP
   - 优点：相比起蓝牙串口 APP，界面更加简洁，操作更方便
   - 控制代码（部分）：

```
    if(ch == 'F'){
         //前进
         Serial.println("up");
         motor_left.setSpeed(210);
         motor_right.setSpeed(180);
         motor_left.run(BACKWARD);
         motor_right.run(BACKWARD);
```

```
        }
        else if(ch == 'B'){
            //后退
            Serial.println("back");
            motor_left.setSpeed(210);
            motor_right.setSpeed(180);
            motor_left.run(FORWARD);
            motor_right.run(FORWARD);
        }
```

4.3. 视频讯号传递

- 使用软件：DroidCamApp
- 阶段结果：成功实现与 PC 端的视频传递
- 传输延迟：USB < WLAN < Bluetooth



连接
(✓ WiFi / LAN)
ADB(由USB)
蓝牙
服务器模式
手机IP: 10.48.18.102
DroidCam端口: 4747
Stop
启用音频
[✓ 启用视频]
镜像网络视频
DroidCam 客户端 v3.2 汉化:GCA机锋汉化组_abc8054

Figure1 在 PC 端显示视频讯号

4.4. 图像处理初步尝试

1. 环境配置
   Python 2.7
   openCV
   Numpy
   Matplotlib

2. 图片相似度算法

   - **直方图**
     在 Python 中利用 opencv 中的 calcHist()方法获取其直方图数据, 返回的结果是一个列表，使用 matplotlib，画出了这两张图的直方图数据图

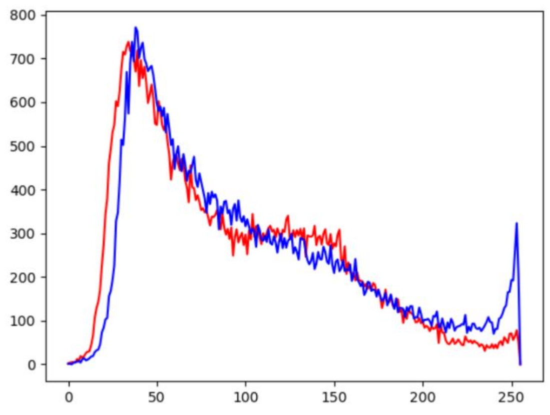


Figure2 直方图示例

- **平均哈希法(aHash)**
  此算法是基于比较灰度图每个像素与平均值来实现的
  一般步骤：
  a. 缩放图片，一般大小为 8*8，64 个像素值
  b. 转化为灰度图
  c. 计算平均值：计算进行灰度处理后图片的所有像素点的平均值
  d. 比较像素灰度值：遍历灰度图每一个像素，大于平均值记录为 1，否则为 0
  e. .得到信息指纹：组合 64 个 bit 位，顺序随意保持一致性
  最后比对两张图片的指纹，获得汉明距离即可。

- **感知哈希算法(pHash)**
  平均哈希算法过于严格，不够精确，更适合搜索缩略图，为了获得更精确的结果可以选择感知哈希算法，它采用的是 DCT（离散余弦变换）来降低频率的方法

- dHash 算法
  相比 pHash，dHash 的速度要快的多，效果要更好，它是基于渐变实现的
  步骤：
  a. 缩小图片：收缩到 9*8 的大小，以便它有 72 的像素点
  b. 转化为灰度图
  c. 获得指纹：如果左边的像素比右边的更亮，则记录为 1，否则为 0.
  最后比对两张图片的指纹，获得汉明距离即可

3. 算法比较

	去除颜色 保留灰度	**绝对位置 设置偏差**	**灰度整体 基准不同**
直方图	0.50784498	0.77417177	0.54242122
改进的直方图	0.48776686	0.79021007	0.48097572
aHash	3	18	11
pHash	6	23	5

4. 结果分析

- 对于不同对比组之间打乱顺序的相似度比较，所有算法的相似度都很低
- 可以看出，对于绝对位置有偏移的图片，直方图算法结果更好
- 对于改变整体灰度或者色调的图片，哈希算法更加有效
- 在之后的视频处理中可以参考借鉴这些方法

4.5. 人员分工

- 赵元培：编写、调试手机控制小车程序
  完成并对比多种处理图片相似度的简单算法
  展示阶段成果

- 胡景文：完善蓝牙串口等硬件组装
  openCV 等的环境搭建
  查阅处理图片相似度的算法

- 杨文清：完善蓝牙串口等硬件组装
  寻找更好的手机端控制 APP
  查阅处理图片相似度的算法

4.6. 存在的不足

- 暂时没有使用 scikit 工具进行机器学习算法的应用
- Python 的视频读入存在问题

5. 第三阶段：深度学习识路

5.1. 阶段目标

- 定义寻路具体问题
- 收集训练、测试集
- 用视频图片处理算法处理训练集
- 利用测试集测试不同深度学习算法
- 选定算法，利用蓝牙传回小车
- 修改 arduino 程序，直到实际测试满意

5.2. 图像处理

- 使用软件：DroidCamApp
- 阶段结果：成功实现与 PC 端的视频传递

5.2.1. 图像识别算法

5.2.1.1. 相似性度量

图像相似度计算主要用于对于两幅图像之间内容的相似程度进行打分，根据分数的高低来判

断图像内容的相近程度。可以用于计算机视觉中的检测跟踪中目标位置的获取，根据已有模板在图像中找到一个与之最接近的区域。还有一方面就是基于图像内容的图像检索，也就是通常说的以图检图。比如给你某一个人在海量的图像数据库中罗列出与之最匹配的一些图像，当然这项技术可能也会这样做，将图像抽象为几个特征值，比如 Trace 变换，图像哈希或者 Sift 特征向量等等，来根据数据库中存得这些特征匹配再返回相应的图像来提高效率。

通过之前的探究，我们最终选定色彩分布直方图进行度量。

5.2.1.2. 色彩分布直方图

利用相关算法，我们实现了识别特定区域的特定颜色的功能，并将其应用在小车的一个实践上：斗牛。小车通过算法检测到红色，便向那里加速冲去，若此时红色突然消失，小车将会进行急转弯；如果屏幕是大面积黑色，也就是模拟被蒙上双眼的情况，此时小车会停止。过程十分类似斗牛，测试效果也十分生动有趣。

5.3. 数据集获取

我们针对设计的不同情况，模拟小车运动视角拍摄视频，并选取共计 1000 张图片

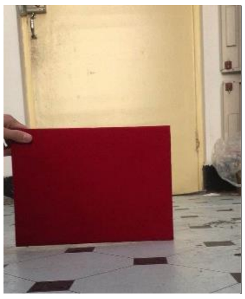


Figure 5-1

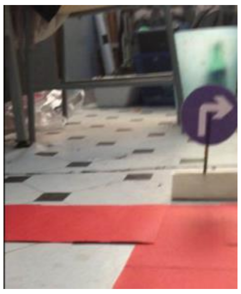


figure 5-2

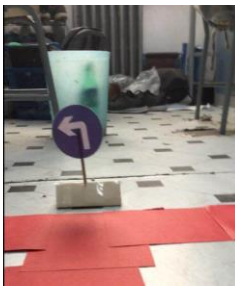


figure 5-3

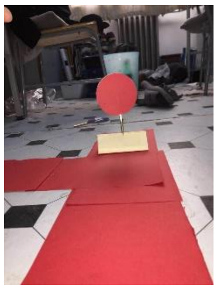


figure 5-4

Figure 5-1：模拟“斗牛”场景的红布，小车发现红布会加速前进

Figure 5-2：右转标识牌（偏右视角），小车会缓慢右转

Figure 5-3：左转标识牌，小车会缓慢左转

Figure 5-4：红灯，小车会停止，直到变为绿灯

5.4. 深度学习算法选取

我们测试了 ANN，CNN，RNN 的测试效果，暂时选定 RNN 作为模型算法

优点：时间递归神经网络可以描述动态时间行为，因为和前馈神经网络接受较特定结构的输入不同，RNN 将状态在自身网络中循环传递，因此可以接受更广泛的时间序列结构输入。

缺点：简单递归神经网络无法处理随着递归，梯度爆炸或者梯度消失的问题，并且难以捕捉长期时间关联；有效的处理方法是忘掉错误的信息，记住正确的信息。LSTM 能够比较好的解决这个问题。

```
cell=tf.nn.rnn_cell.BasicLSTMCell(10)
unstack_x = tf.unstack(x, 10, 1)
lstm_cell = rnn.BasicLSTMCell(10, forget_bias=1.0)
outputs, states = rnn.static_rnn(lstm_cell, unstack_x, dtype=tf.float64)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    outputs_array=(sess.run(outputs,feed_dict={x:train_x}))
```

5.5. 实现效果

5.5.1. 斗牛

可以实现小车通过算法检测到红色，向那里加速冲去，若此时红色突然消失，小车将会进行急转弯；如果屏幕是大面积黑色，也就是模拟被蒙上双眼的情况，此时小车会停止。

5.5.2. 识路

- 实现了小车在慢速状态下识别 90 度左转和 90 度右转
- 实现了小车在 T 字路口根据转向标志判断左转右转
- 实现了小车在直线道路上遇到红灯停止，等待红灯变为绿灯再次前进

5.6. 人员分工

- 赵元培：探讨最终实现形式
  视频预处理
  探究深度学习算法（RNN）
  深度学习代码具体实现
  Arduino 代码具体实现
  蓝牙重定位

- 胡景文：探讨最终实现形式
  收集数据集

探究深度学习算法（CNN）
测试实际情况，修改算法

- 杨文清：探讨最终实现形式
  收集数据集
  探究深度学习算法（ANN）
  图像识别算法调研
  测试实际情况，修改算法

5.7. 存在的不足

- 只实现了在和测试集十分相同的路况下的情形，小车可能无法判断更为复杂的情况
- 小车移动速度较为缓慢，有待提高
- 道路选为红色，和红灯颜色相近，导致"闯红灯"现象经常发生
- 深度学习算法中的各项参数调节不是很深入，可能没有达到最好效果

6. 总结与展望

在这个项目中，我作为组长担任了完成项目的主力，基本所有代码实现都是我完成的，我在实现代码的过程中对于各种算法有了进一步的认识；通过了解 arduino，我提高了对于陌生硬件的学习能力；同时作为组长，我也锻炼了我的组织协调能力和领导能力。

通过这个一学期的项目，我熟悉并掌握了 arduino 的基础用法；将课堂上学习到的深度学习算法进行了实践，并在实践中通过不断解决问题提高了自己的系统工程学能力。

关于这个项目的展望，我希望能以此为契机，不断完善该项目，使其具有一定的使用价值。同时对于我个人，我也希望能在未来对于人工智能进行深入研究，在这条路上踽踽独行。

课程学习笔记

本学期我们以《Hands On Machine Learning》一书为指导，从机器学习的基础（回归、分类问题；损失函数的确定；精确率和召回率；训练集和测试集的选取等等）开始，到常见的机器学习算法（KNN 模型；线性回归模型；支持向量机模型；决策树模型等等），最后讲到深度学习的算法分析（卷积神经网络；循环神经网络等等）。

作为人工智能的基础课，我从课上学习到了很多，从基础的概念到神经网络的深入分析，我认为作为计算机专业的学生，决不能仅仅只是会调包而已，而是要深入了解、熟悉并掌握常用的深度学习算法，这样才有利于我们未来的发展。

在本学期中，我在课程中对讲授内容进行了整理，在梳理课程脉络的同时也记录了不少自己的思考和体会，以下便是我这一学期的课堂笔记整理，部分文字和图片来自于课程 PPT。我也在课后完成了大部分实例代码的运行，都得到了和预期一致的结果，由于篇幅所限，便不将代码运行结果列出了。

CHAPTER 1 - INTRODUCTION

Machine learning is the science of programming computers so they can learn from data.

Some of the supervised learning algorithms

- K-nearest neighbors
- Linear regression
- Logistic regression
- Support vector machines
- Decision trees and random forests
- Neural networks

Some of the unsupervised learning algorithms

- Clustering
- Visualization and dimensionality reduction
- Association rule learning

Instance-based learning
Model-based learning

Choose data:

- Avoid nonrepresentative training data
- Avoid irrelevant features
- Avoid overfitting
- Avoid underfitting

Train and test

- Cross-validation : split the training set into subsets, each model is trained against a different combination of these subsets and validated against the remaining parts
- Hyperparameters

CHAPTER 2 - REGRESSION

Take the California Housing Prices dataset as example

Frame the problem

- Supervised learning
- Regression task
- Batch learning

Select a performance measure

- RMSE

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

- MAE

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(x^{(i)}\right) - y^{(i)}\right|$$

- L_k norm

$$\|v\|_k = (|v_0|^k + |v_1|^k + \cdots + |v_n|^k)^{\frac{1}{k}}$$

Choose feature

- One-hot
- Normalization and standardization
  -Machine Learning algorithms don't perform well when the input numerical attributes have very different scales.

Select and train a model

Find out what you want to do, **classification** or **regression**
Take decision tree as example:

- Split the data into train set and test set
- Cross-Validation
  -split the training set into a smaller training set and a **validation set**
  -use **cross-validation feature**(Scikit-Learn)
- Try different combination of hyperparameter values
  -use **GridSearchCV**(Scikit-Learn)
  --try out all possible combinations
  --is fine exploring relatively few combinations

--the hyperparameter search space is large
-use **RandomizedSearchCV**(Scikit-Learn)
--evaluates a given number of random combinations by selecting a random value **for each hyperparameter at every iteration**
--take the iteration into consideration
--have more control over the computing budget

Evaluate on the Test Set

- get the predictors and the labels from test set
- run your full_pipeline to transform the data
- evaluate the final model on the test set

Launch, Monitor, and Maintain Your System

CHAPTER 3 - CLASSIFICATION

using the MNIST dataset as example (handwritten recognition)

Shuffle the training set

- guarantee that all cross-validation folds will be similar
- some learning algorithms are sensitive to the order of the training instances

Training a Binary Classifier

- "is" or "not"
- **Stochastic Gradient Descent (SGD) classifier**
  -is capable of handling very large datasets efficiently

Performance Measures

- **K-fold cross-validation**
  -splitting the training set into K-folds
  -making predictions
  -evaluating them on each fold using a model trained on the remaining folds
- **Confusion Matrix**
  -count the number of times instances of class A are classified as class B

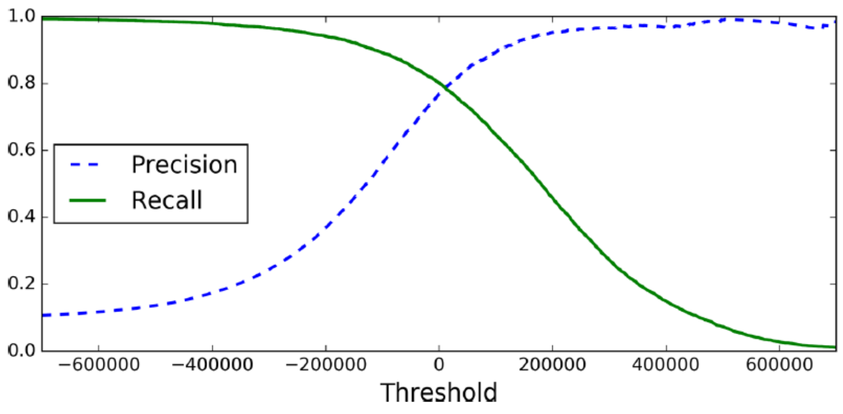
true negatives	***false positives***
false negatives	***true positives***

$$precision = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

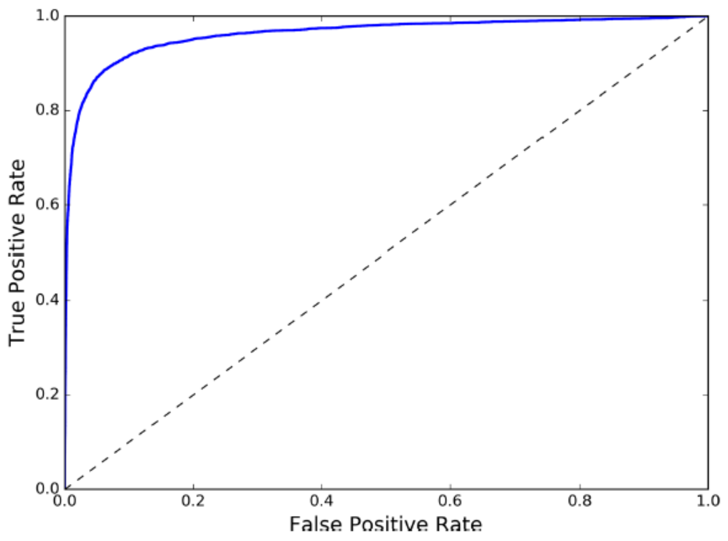
$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

- **precision/recall tradeoff**



Set the threshold to decide the preference of precision/recall

- **ROC Curve**



Multiclass Classification

distinguish between more than two classes
Random Forest classifiers and **naive Bayes classifiers**

- **one-versus-one** (OvO) strategy
  -train a binary classifier for every pair of digits
  -need to train $N \times (N-1) / 2$ classifiers
- **one-versus-all** (OvA) strategy
  -for each class, train a binary classifier to classify if data belongs to this class
  -need to train N classifiers
  -is preferred

Error Analysis of multiclass classification

- **confusion matrix**
  -gain insights on ways to improve your classifier
  -gain insights on what your classifier is doing and why it is failing

Multilabel Classification

output multiple classes for each instance

- **KNeighborsClassifier**
  -measure the **F1 score** for each individual label

Multioutput Classification

a system that removes noise from images as example

CHAPTER 4 – TRAINING SIMPLE MODELS

Linear Regression

- computing a weighted sum of the input features
- plus a constant called the *bias term*

- $\hat{y} = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \cdots + \Theta_n x_n$

  -$\hat{y}$ is the predicted value.
  -n is the number of features.
  -x_i is the ith feature value.
  -Θ_j is the jth model parameter

- $\hat{y} = h_\Theta(x) = \Theta^T \boldsymbol{x}$

  -θ is the model's *parameter vector*, containing the bias term Θ_0 and the feature weights Θ_1 to Θ_n.
  -Θ^T is the transpose of θ (a row vector instead of a column vector).
  -x is the instance's *feature vector*, containing *x*0 to *xn*, with *x*0 always equal to 1.
  -$\Theta^T \boldsymbol{x}$ is the dot product of θ^T and **x**.
  -$h_\Theta(x)$ is the hypothesis function, using the model parameters θ

- $\text{MSE}(\mathbf{X}, h_\Theta) = \frac{1}{m}\sum_{i=1}^{m}(\Theta^T \boldsymbol{x}^{(i)} - \boldsymbol{y}^{(i)})^2$
- **The Normal Equation**

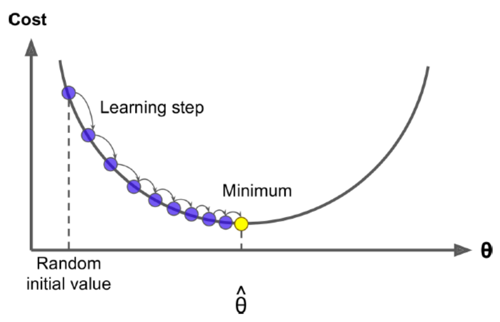
  $\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$

  -the **computational complexity** of inverting matrix is typically about O(n2.4) to O(n3)
  -handle large training sets efficiently, provided they can fit in memory

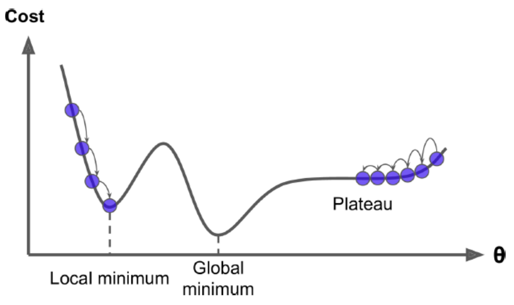
Gradient Descent

to tweak parameters iteratively in order to minimize a cost function

- **Descriptions**
  -start by filling θ with random values
  -improve it gradually, taking one step at a time, attempting to decrease cost function
  -until the algorithm **converges**



- **learning rate**
  - the size of the steps
  - too small : it will take a long time
  - too high : jump across the valley and end up on the other side
  - holes, ridges : cause **local minimum**



-MSE cost function is a **convex function** : there are no local minima

Batch Gradient Descent

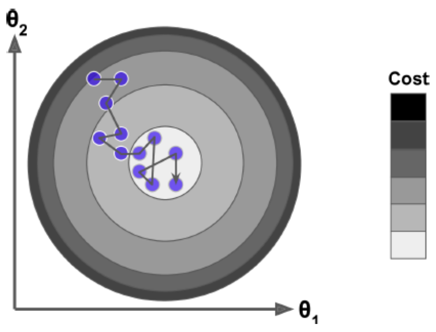
to compute the gradient of the cost function with regards to each model parameter Θ_j

$$\nabla_\theta \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

- terribly slow on very large training sets
- scales well with the number of features

Stochastic Gradient Descent

- picks a random instance in the training set at every step
- computes the gradients based only on that single instance
- much less regular than Batch Gradient Descent



Mini-batch Gradient Descent

- computes the gradients on small random sets of instances called **mini-batches**
- you can get a performance boost from hardware optimization of matrix operations

Polynomial Regression
use a linear model to fit nonlinear data

- add powers of each feature as new features
- train a linear model on this extended set of features

The Bias/Variance Tradeoff

- **Bias**
  due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.

- **Variance**
  due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance, and thus to overfit the training data.

- **Irreducible error**
  This part is due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data

Ridge Regression

a regularized version of Linear Regression: a *regularization term* equal to $\alpha \sum_{i=1}^{n} \Theta_i^2$ is added to the cost function

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

- use L2 norm
- α controls how much you want to regularize the model

Lasso Regression

add a regularization term to the cost function

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

- use L1 norm
- tend to completely eliminate the weights of the least important features

Elastic Net

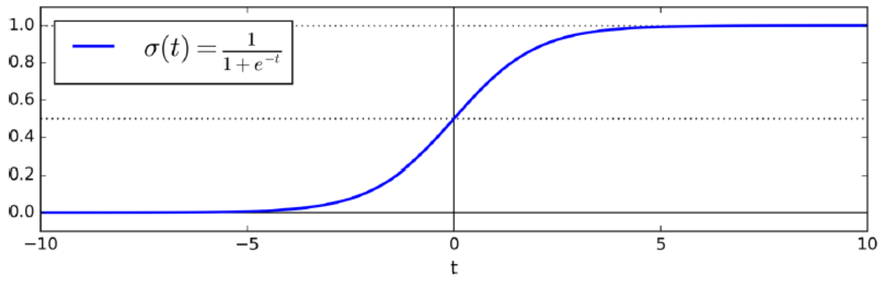
a middle ground between Ridge Regression and Lasso Regression
(actually a mixture)

$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2$$

Logistic Regression

instead of outputting the result directly, it outputs the **logistic** of this result

$$\hat{p} = h_\theta(\mathbf{x}) = \sigma\left(\theta^T \cdot \mathbf{x}\right)$$



- Estimating Probabilities
- Training and Cost Function

Softmax Regression

- computes a score $s_k(\mathbf{x})$ for each class k
- estimates the probability of each class by applying the **softmax function** to the scores

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

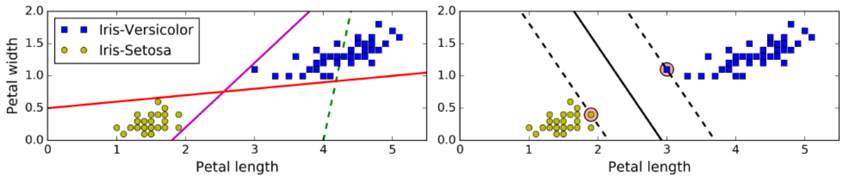
CHAPTER 5 – SVM

A **Support Vector Machine (SVM)** is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection

particularly well suited for classification of **complex but small- or medium-sized datasets**

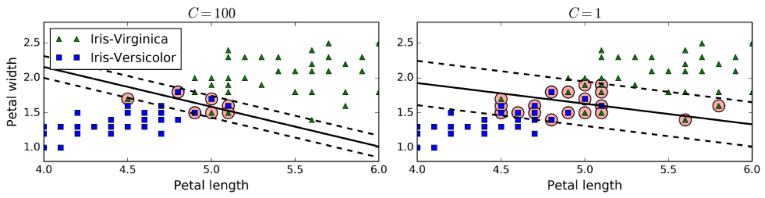
Linear SVM Classification

- an SVM classifier as fitting the widest possible street between the classes
- This is called **large margin classification**
- sensitive to the feature scales

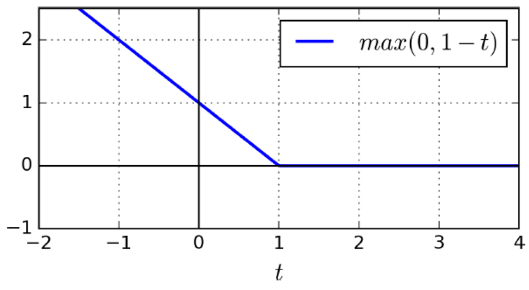


Soft Margin Classification

- find a good balance between keeping the street as large as possible and limiting the **margin violations**
- **C hyperparameter** : a smaller C value leads to a wider street but more margin violations



Hinge Loss



SVM Regression

instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible *on* the street while limiting margin violations

CHAPTER 6 – DECISION TREES

Making Predictions

- Node's **samples** attribute counts how many training instances it applies to
- Node's **value** attribute tells how many train instances of each class this node applies to
- **gini score**

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

Estimating Class Probabilities

- traverse the tree to find the leaf node for this instance
- return the ratio of training instances of class *k* in this node

Training Algorithm

- splits the training set in two subsets using a single feature *k* and a threshold t_k
- splits the subsets using the same logic, then the sub-subsets and so on, recursively
- stops recursing once it reaches the maximum depth

Regularization Hyperparameters

- to avoid overfitting the training data
- restrict the **maximum depth** of the Decision Tree
- contril the minimum number of samples a node must have before it can be split
- contril the minimum number of samples a leaf node must have

Regression

- tries to split the training set in a way that minimizes the MSE
- prone to overfitting when dealing with regression tasks

Instability

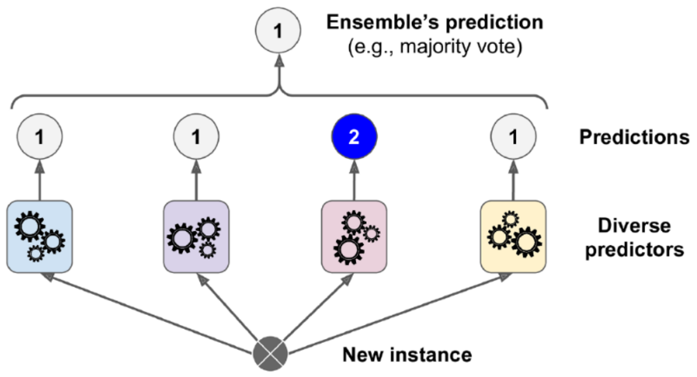
- sensitive to training set rotation
- sensitive to small variations in the training data

CHAPTER 7 – RANDOM FORESTS

aggregate the predictions of a group of predictors
This technique is called **Ensemble Learning**

Ensemble methods

- **Voting Classifiers**
  -aggregate the predictions of each classifier
  -predict the class that gets the most votes
  -work best when the predictors are as independent from one another as possible



- **Bagging and Pasting**
  -use the same training algorithm for every predictor, but to train them on different random subsets of the training set
  -**bagging** : sampling is performed *with* replacement
  -**pasting** : sampling is performed *without* replacement

Random Forests

- an ensemble of Decision Trees
- generally trained via the bagging method
- typically with **max_samples** set to the size of the training set

Boosting

Ensemble method that can combine several weak learners into a strong learner

- **AdaBoost**
  -pay a bit more attention to the training instances that the predecessor underfitted
  -**weighted error rate**

$$r_j = \frac{\displaystyle\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\displaystyle\sum_{i=1}^{m} w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j} \qquad w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$

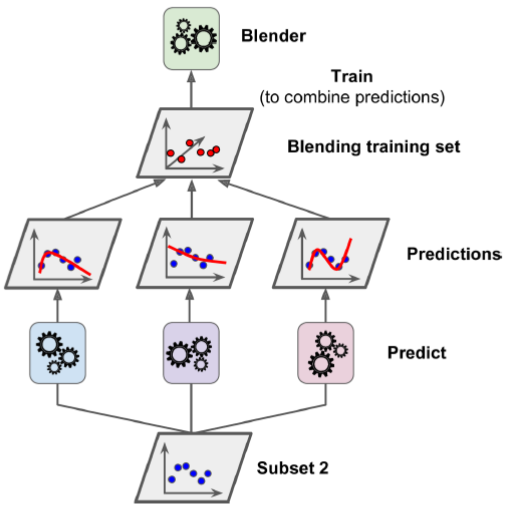
-a new predictor is trained using the updated weights, and the whole process is repeated
-to make predictions, AdaBoost simply computes the predictions of all the predictors and weighs them using the predictor weights *αj*

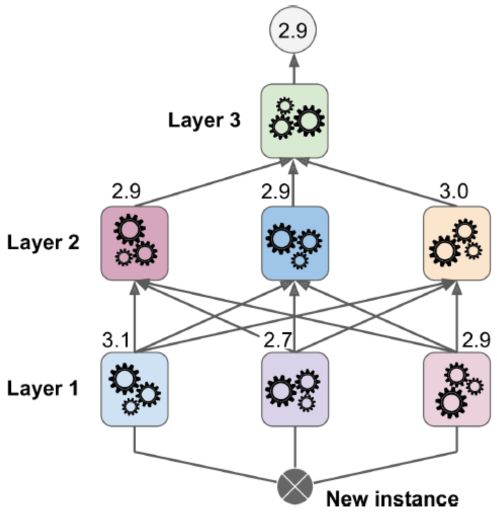
- **Gradient Boosting**

  -fit the new predictor to the *residual errors* made by the previous predictor
  -support a **subsample** hyperparameter
  --trade a higher bias for a lower variance
  --speed up training considerably

Stacking

train a model to perform the aggregation

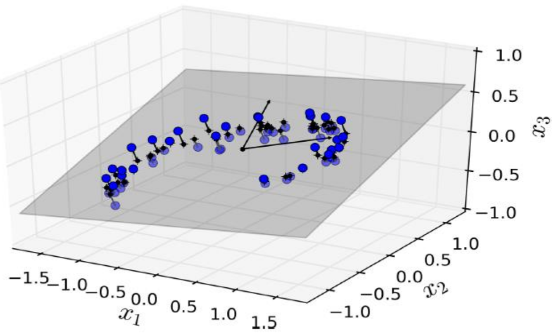




CHAPTER 8 – DIMENSIONALITY REDUCTION

Projection

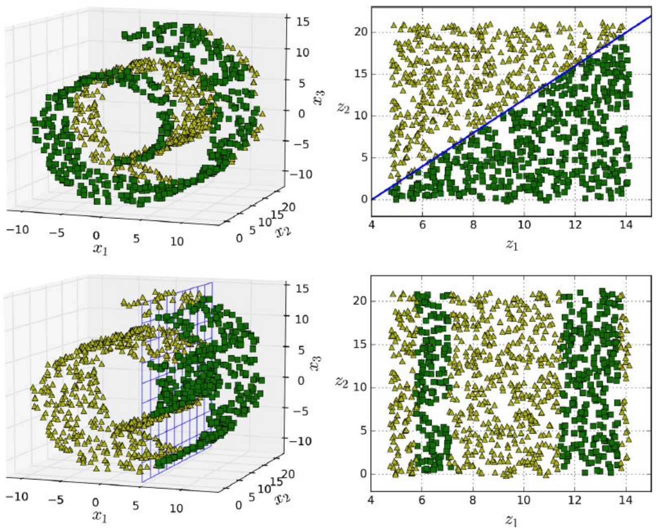
Many features are almost constant, while others are highly correlated



this is a lower-dimensional (2D) subspace of the high-dimensional (3D) space
if we project every instance perpendicularly onto this subspace, we get the new 2D dataset

Manifold Learning

model the **manifold** on which the training instances lie



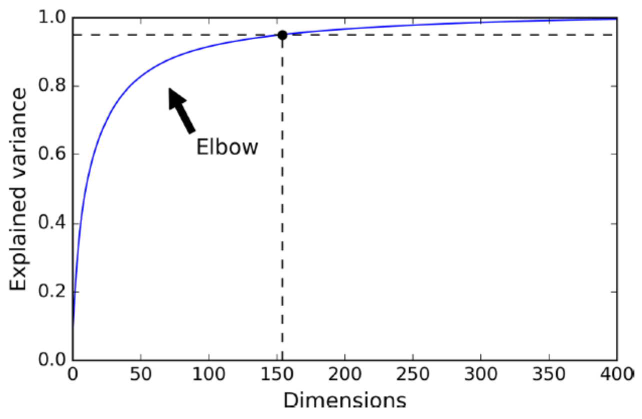
PCA

- identifies the hyperplane that lies closest to the data
- projects the data onto it

Choosing the Right Number of Dimensions

- choose the number of dimensions that add up to a sufficiently large portion of the variance

- plot the explained variance as a function of the number of dimensions



LLE

Locally Linear Embedding

- measuring how each training instance linearly relates to its closest neighbors
- looking for a low-dimensional representation of the training set where these local relationships are best preserved

CHAPTER 9 – TENSORFLOW

a powerful open source software library for numerical computation, particularly well suited and fine-tuned for large-scale Machine Learning

- the first part builds a computation graph
  -**construction phase**
  -builds a computation graph representing the ML model
  -the computations required to train it
- the second part runs it
  -**execution phase**
  -generally runs a loop that evaluates a training step repeatedly

Lifecycle of a Node Value

- All node values are dropped between graph runs, except variable values, which are maintained by the session across graph runs
- In single-process TensorFlow, multiple sessions do not share any state, even if they reuse the same graph

Linear Regression with TensorFlow

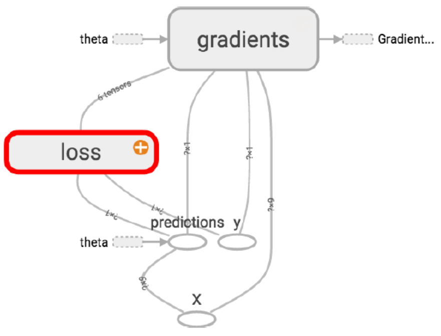
- The **input** and **output** can be of any number
- tensors are simply represented by **NumPy** ndarrays

Saving and Restoring Models

- instead of initializing the variables using the init node, call the restore() method
- By default a Saver saves and restores all variables under their own name, but you can specify which variables to save or restore

Name Scopes

with tf.name_scope("loss") as scope:

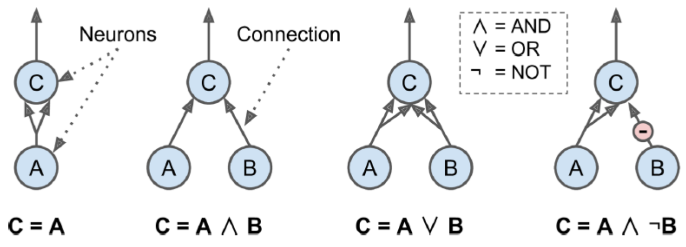


CHAPTER 10 – ARTIFICIAL NEURAL NETWORKS

core of Deep Learning, versatile, powerful, and scalable

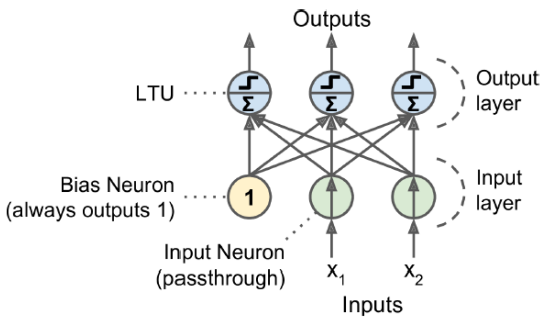
Logical Computations with Neurons

- one or more binary (on/off) inputs and one binary output
- simply activates its output when more than a certain number of its inputs are active



The Perceptron

- the inputs and output are now **numbers**
- each input connection is associated with a weight
- The LTU computes a weighted sum of its inputs
- applies a step function to that sum
- be used for **simple linear binary classification**



- **Hebb's rule** : when a biological neuron often triggers another neuron, the connection between these two neurons grows stronger

Multi-Layer Perceptron and Backpropagation

- one input layer
- one or more layers of LTUs, called **hidden layers**
- one final layer of LTUs called the **output layer**
- **backpropagation** training algorithm
  -makes a prediction **(forward pass)**
  -measures the error
  -goes through each layer to measure the error from each connection **(reverse pass)**
  -slightly tweaks the connection weights to reduce the error **(Gradient Descent step)**
- replaced the step function with the **logistic function**
  -**The hyperbolic tangent function** $\tanh(z) = 2\sigma(2z) - 1$
  -**The ReLU function** $\mathrm{ReLU}(z) = \max(0, z)$.

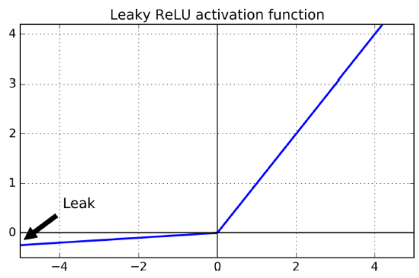
CHAPTER 11 – DEEP NEURAL NETS

Vanishing/Exploding Gradients Problems

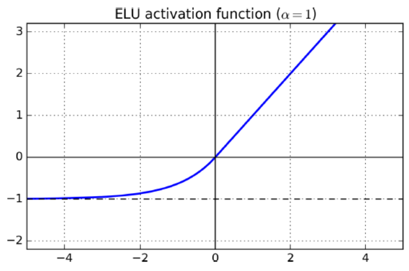
- **vanishing gradients problem**
  -gradients often get smaller as the algorithm progresses down to the lower layers
  -As a result, the Gradient Descent update leaves the lower layer connection weights virtually unchanged, and training never converges to a good solution
- **exploding gradients problem**
  -the gradients can grow bigger and bigger
  -many layers get insanely large weight updates and the algorithm diverges

Nonsaturating Activation Functions

- **leaky ReLU**



  -The hyperparameter α defines how much the function "leaks"
- **ELU**
  -exponential linear unit



Gradient Clipping

- to simply clip the gradients during backpropagation
- mostly useful for recurrent neural networks

Unsupervised Pretraining

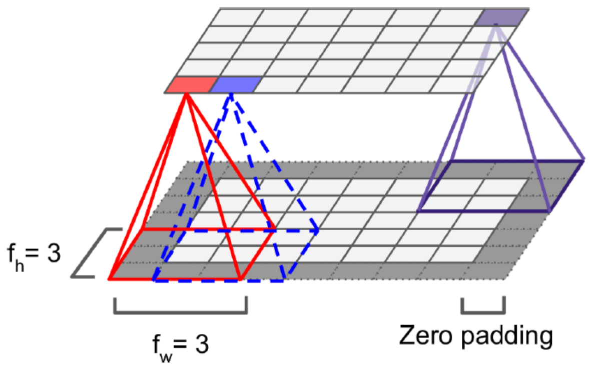
- you have plenty of unlabeled training data
- train the layers one by one
- start with the lowest layer and go up, using an unsupervised feature detector algorithm
- all layers were trained this way, you can finetune the network using supervised learning

CHAPTER 13 – CONVOLUTIONAL NEURAL NETWORKS

achieve superhuman performance on some complex visual tasks

Convolutional Layer

- neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields
- **zero padding**



- **Filters**

  -an representative mark of a field

  -a layer full of neurons using the same filter gives you a **feature map**

Stacking Multiple Feature Maps

- all neurons located in the same row i and column j but in different feature maps are connected to the outputs of the exact same neurons in the previous layer
- compute the output of a given neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f_{n'}} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = u \,.\, s_h + f_h - 1 \\ j' = v \,.\, s_w + f_w - 1 \end{cases}$$

Pooling Layer

- their goal is to *subsample (i.e., shrink) the input image in order to* reduce the computational load, the memory usage, and the number of parameters
- you must define its size, the stride, and the padding type
- a pooling neuron has no weights
- all it does is aggregate the inputs using an aggregation function such as the max or mean