人工智能基础（高中版）

辅导讲义

王斐 Michael[1]

School of Mathematics and Statistics, UCD

E-mail: michael.yunfei@gmail.com

[1] *Author Website*

Contents

第三章: 神经网络模型初探

亲爱的同学们，从这一章开始我们将会接触目前比较流行的人工智能模型-神经网络模型。相较前一章节，本章难度有所提升，而且模型背后的相关概念更加抽象。如果你已经开始阅读第三章节的内容，或许你会觉得 '不知所云' 或者 '无从下手'。对于任何一个初学者来说，这都是很正常的经历，希望你们不要气馁，更不要怀疑自己。此时此刻，希望同学们仍然要怀着探索的心态去进入这一章节的学习。

《大学》中语: 知止而后有定，定而后能静，静而后能安，安而后能虑，虑而后能得。物有本末，事有终始。知所先后，则近道矣。

我们将会在接下来的三周里沉浸在神经网络模型中，如果同学们紧跟老师的节奏，按照要求完成**课前预习，课上笔记，及课后练习**这三个环节，我可以向你们保证，三周之后你们（每一位同学）完全可以:

- 理解为什么人工智能在神经网络[1]出现后具有了广泛得应用价值;
- 掌握神经网络的本质所在，并且了解模型背后的思想精髓所在;
- 能够用 Python 透过向量[2]和矩阵[3]进行编写 20 行左右的小程序，从而以此去理解神经网络的模型设计逻辑;
- 能够使用 Python 中人工智能学习平台, 如 TensorFlow 来进行神经网络的训练和调试，从而可以独立自主得对大量的图片数据进行分析。

[1] *Neutral Network*

[2] *Vector*

[3] *Matrix*

为了提高沟通速率且帮助同学们养成良好学习节奏[4]，我们以后会将课前预习，课上笔记，及课后练习简称为 C1, C2, C3。比如，如果我说 '你需要在下周一之前完成 *C1*', 那就意味着你需要根据我在 *C1* 中的指示进行课前预习。下面的表格是后面课程中 C1, C2, C3 中的包含的内容以及所设定的预订目标。

[4] 我大学有个德国的同学曾经自恃聪明，声称自己不需要读任何诗歌就可以作诗，我说你来写一首中文诗歌，他也只能相对无言。每个人的大脑都可以是一个储存器，没有输入就不会有输出，所以 '自学成才' 的前提是自学。不过老师可以负责任的告诉你，自学你需要花更多时间。独孤很可能会败求，切勿舍近求远。

	C1	C2	C3
主题	背景资料阅览	课堂老师讲解	课后习题和编程
内容	网络文章和视频	辅导讲义和课件	数学练习和 Python 练习
时长	30 分钟-1 小时	45 分钟	2-3 小时
频率	一周一次	一周三次	一周一次
收获	✓	只完成 C1 好比看了一场有关 AI 的电影	
	✓	✓	不作习题 C2 完全是浪费时间
	✓	✓	✓

3.1 课前预习 C1

按照惯例，本次 C1 依然有两部分组成:

1. 读网络文章, 看网络视频;
2. 根据指示完成小习题，并且准备在课上使用。

第一部分: 神经网络追本溯源

人们对人工智能的研究热情由来已久，其中神经网络模型的主要研究兴起于上世纪八十年代。众多研究者中，领军人物包括但不限于:Geoffrey Hinton[5], Yoshua Bengio and Yann LeCun。他们三位因此也被称为人工智能三教父[6]。我们以 Geoffrey Hinton 的思路为切入点，来理解神经网络的发展过程和之所以蓬勃发展的深层次原因。

[5] 杰弗里 · 辛顿

[6] *Godfathers of AI (Godfathers of Deep Learning)*

观看 Geoffrey Hinton 的采访视频，注意思考以下几个问题:

- 观看前 10 分钟[7]
- 是什么触发了 Geoffrey Hinton 对人工智能的研究，最初的切入点是什么?
- Geoffrey Hinton 比较了英国和美国的学术环境，你从中有什么启发?
- Geoffrey Hinton 提到了心理学界和 AI 领域对知识的理解，何谓知识?
- 视频连接:https://www.bilibili.com/video/av69581590/

[7] 注意视频中配有机器自动加注的英文字母和中文翻译, 因此存在一定纰漏, 但是不影响整体理解. 其中 *03:50* 处，他讲到 *'I think around early 1982...'*，而不是 '伊拉克发生了什么'。这说明机器和人类一样并不完美，'绝对' 的物只在意念中存有!

哲学延伸

认识论是哲学研究范畴中比较重要的一枝，视频中 Geoffrey 提到 'how concepts are relate to other concepts' (一个概念与其它概念的联系)，这是目前有关知识的普遍共识。比如描述一朵花，需要高度，大小，颜色，等等一系列的概念堆积。而这些概念的关联就构成了我们的知识。**后面我们介绍深度学习模型时，你会接触到 '卷积' 和 '池化' 等概念**。这些新鲜的名字背后本质上是通过一个个数学模型来模拟概念关联和堆积，从而进行智能判断。深度神经学习本质上是对人的**生物仿生模拟**，我们在调试模型时需要反向调整参数，这与我们实际生活中的学习如出一辙。比如，你第一次接触滚烫的热水时，会感到疼痛，接受到该信号后，下一次你会**试探性**得调整动作去接触热水，这就是人类反向调整的过程。

Geoffrey 能够想出神经学习模型除了与他自己兴趣广泛，对该问题长期关注之外，还与其在英国接受的教育传统有关。认识论的鼻祖可以追溯到笛卡尔 (也就是发明 XY 坐标的那个法国人)，但是第一次系统提出 '所有的知识都是关联' 的是英国哲学家休谟，休谟又启发了德国的大哲学家康德。人类历史上几次**颠覆**性的概念，如牛顿万有引力，达尔文进化论，休谟的认识论，亚当斯密的市场经济理论，图灵的计算边界论等都首先出自英国，是因为长期以来英国教育注重热爱自然，着重培养学生仔细观察记录和思考有关。同学们正值青春年华，不要泯灭了对自然和生命得热爱。以后人工智能可以帮助人类去完成更多机械化的任务，那么想象力就变得更加重要。

观看华为总裁任正非的采访视频，注意思考以下几个问题:

- 观看视频中 10:00 - 15:00 分钟段
- 为什么任正非说人工智能是 '计算机 + 统计学'?
- 为什么有一种论调称 '数据就是未来的石油'?
- 视频连接:https://m.sohu.com/a/290453977_652527/

打破人工智能的迷思

一段时间以来，有关人工智能, 大数据的话题占据了很多网络报刊的头条位置，也使得我们的生活有所喧嚣，很多人也是人云亦云般得吹泡泡。在从大得方向上掌握了，深度神经网络学习模型实际上是对人的生物仿生模拟之后，我们会通过本章的学习，解答以上两个问题。简单讲，一个人的成长需要千锤百炼，人工智能也需要学习，不过人类是通过眼耳鼻喉舌五官来进行感觉输入和经验收集，最终进行意识加工和输出，然而电脑喝得是数据，吐出来的是 '牛奶'. 有朝一日，人工智能很有可能会 '横眉冷对千夫指，俯首甘为孺子牛'。让我们共同期待吧!

第二部分: 课前小习题[8]

课前小习题是为了帮助同学们熟悉机器的思考方式，并且能够理解背后的数学算法。简而言之，机器或者电脑的思考方式就是 '一步一脚印' 的方式，也就是具体问题机械分解化，机械分解数学化，数学运算代数化，代数问题向量/矩阵化。这些小习题都非常简单，但是 '法力无边'。

Question C1-Q1. 计算机又被称为电脑, 电脑的核心是芯片，芯片的计算单元是晶体管。晶体管通电可以表示一种状态，称之为 1，晶体管断电可以表示一种状态，称之为 0。假设我们有下列三个并排晶体管, 每个晶体管中可以填 0 或者 1。问三个晶体管中至多可以存放多少个信息 (只要数字不同就可以被成为信息，比如 0 跟 101 不同，那么这两个可以算两个信息。1 和 1 只能算作一个信息)?

Question C1-Q2. 将所有可能的信息 (实际上就是由 0 和 1 组成的序列) 按照你认为符合逻辑的方式进行排列，比如 $0, 1, 00, 01, \cdots$ 或者 $010, 000, 01, 00, \cdots$.

Question C1-Q3. 可能同学们在第一小问中很快就可以得出答案，共计有 $2^3 = 8$ 个信息单元，但是如果你把所有的可能都写出后，你会发现答案是 $2 + 2^2 + 2^3 = 14$.

[8] 勿以善小而不为: 比如现在火爆的 5G 网络, 归根结底起源于是对一个多项式的求解。5G 是错误纠正码的一种 (error correction code), 而错误纠正码中传输速率最大的一种模型依靠的是伽罗瓦理论 (Galois theory), 而这位法国的年轻人提出这个理论最初是解决下面这个问题: 取任意 $a, b, c, d, e, f \in \mathbb{R}$(实数 $\mathbb{R}$ 包括有理数和无理数), 下面五项式方程是否有解,

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$$

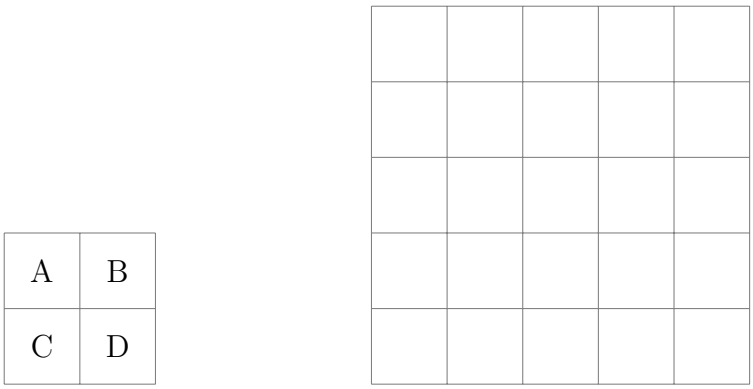
同学们肯定都知道如何解决二项式 $ax^2 + bx + c = 0$ 是否有解, 有兴趣也可以把解决四项式或者五项式作为爱好培养。Galois 只是感兴趣而研究这个问题, 但是该问题所引发的一般理论成为了现代代数的的核心，广泛应用信息传输中。Galois 为博取一女生的欢心，在与情敌的决斗中受重伤，于英年 20 岁时卒。法国政府将其名字刻在 Eiffel Tower 上，以表尊缅。

Remark. 如果我告诉你以下数字的二进制表达，你能否根据以上习题反向推出二进制的设计机制?

十进制数字	二进制数字
14	1110
13	1101
12	1100
11	1011

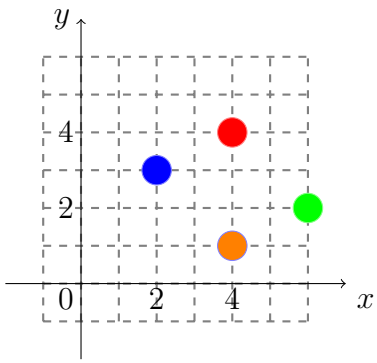
Question C1-Q4. (排列组合题) 现在有以下 7 个空位，如果有三个捆绑在一起的信息 ABC(排序不可改变), 且每个字母占位一个空格，请问至多有几种方式将 ABC 放到下面的方格中去?

Question C1-Q4. (排列组合题) 现在有以下一个 2×2 和一个 5×5 的矩阵, 其中 2×2 的矩阵中已经被填充了 A, B, C, D（如图所示），请问按照现有 A, B, C, D 的排列方式，在 5×5 的矩阵中，至多有多少种填充方式?



Question C1-Q5. (颜色填充) 下图左边的坐标中定位了红，绿，蓝三个点和一个橙色点[9]，如果我们将矩阵的纵列 (row) 代表坐标的 x 轴，将矩阵的横列 (columns) 代表 y 轴，根据所在的坐标将这三个点定位到右边的矩阵去中。我们使用数字代码 1 代表红色，2 代表绿色，3 代表蓝色。

[9] 如果同学有颜色识别困难的，可以问下父母或者身边的朋友。



	y1	y2	y3	y4	y5
x1					
x2			3		
x3					
x4				1	
x5		2			
x6					

因为除红、绿、蓝三色之外的颜色都可以由这三个颜色来调和，因此图中橙色点可以用 $1, 2, 3$ 的组合来代表。[10]

Question C1-Q6. (简单向量计算) 请根据提示完成下列向量和矩阵的计算:

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}2\\6\\9\end{bmatrix} = \begin{bmatrix}3\\8\\\end{bmatrix}; \qquad \begin{bmatrix}1 & 3\\5 & 6\end{bmatrix} + \begin{bmatrix}2 & 7\\6 & 8\end{bmatrix} = \begin{bmatrix}3\\11 & 14\end{bmatrix}$$

Question C1-Q7. (Python 编程题) 阅读下列代码，并且复制到你自己的 Python 平台上运行且查看结果。以下代码是我们 C1-Q1 到 C1-Q6，问题所涉及概念的程序演练[11]。

```
# Chapter 3 Pre-lecture Exercise
# 第三章课前练习
# @ Michael

# Import essential packages
# 导入必要扩展

import numpy as np

# Create a vector and matrix, 创建向量和矩阵

a = np.array([1, 2, 3])
print(a)
type(a)
a.dtype
a.ndim
a.shape

a1 = np.array([4, 5, 6])
a2 = np.array([7, 8, 9])
```

[10] 因为颜色还有浓艳的问题，所以标准的红、绿、蓝，也就是速成的 *RGB* 颜色代码是用 *255* 来表示浓度均值的。比如红色的代码是 *(255, 0, 0)*，绿色为 *(0, 255, 0)*，蓝色为 *(0, 0, 255)*。比如老师个人很喜欢的法式浅蓝色的代码是 *(70, 136, 241)*。

[11] *Numpy* 是 *Python* 中常用的数据处理扩展程序，其全称为 *Numberical Python* (数值计算的 *Python*)。需要提醒同学们的是。*Python* 语言的发展是一个不断融合其它编程语言的过程。其中，对其影响最大的几个有：*C, C++, Java; Matlab; R*. 其中第一类主要是对 *Python* 的性能的语言结构上的影响。第二类 *Matlab* 是对其数值计算和可视化的影响（你甚至可以说,*Numpy+Matlplot* = *Matlab*)。比如向量的建立在 *Matlab* 中可以直接输入 *[1, 2, 3]*，*Python* 的输入仿照了 *Matlab*，再比如 *np.zeros([3, 3])* 就是直接复制 *Matlab*。另外，*Python* 中常用的可视化工具 *Matplotlib*，全称是 *Matlab Plot Library*，名字就不言而喻了。因为 *R* 语言在统计和数据处理上有广泛得应用，所以 *Python* 的 *pandas* 这个扩展程序主要是借用了 *R* 语言中 *dataframe* 这个概念。

```
print(a1+a2)  # 简单的矢量计算

b = np.array([1, 2, 3], [5.6, 7.8, 9.9])  # 错误的输入
b = np.array([[1, 2, 3], [5.6, 7.8, 9.9]])
b.dtype
b.ndim
b.shape  # always use shape to check the dimension
print(b)

b1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
b1.shape
b2.shape
print(b1+b2)

np.ones([3, 3])  # 快速创建3x3的0矩阵
np.zeros([6, 6])  # 快速创建6x6的1矩阵
```

C1-Q1 的程序化解决[12]

```
# C1-Q1 的程序化解决

def comper(n, q):
    # A function for calculating all possible combinations of sequences
    # Input:
    # n - maximial length of sequence
    # q - number of digits, e.g., 2 means binary, 3 means ternary
    # Output: number of all possible sequences we can get given n and q
    # 计算所有可能数组
    # 输入:
    # n - 数组的长度的最大值
    # q - 可选数值，比如2 意味着二进制可选0， 1； 3意味着可选0, 1, 2
    # 输出: 所有可能数组的数量
    t = 0  # initialize a variable to store the result
    # 初始化一个变量t,用来储存结果
    for i in range(n):
        temp = pow(q, i+1)
        t = t + temp
    return(t)


# Test our function, 测试我们的方程
result = comper(3, 2)
print(result)
# Try different parameters
comper(32, 2)  # 8589934590
comper(64, 2)  # 36893488147419103230
comper(32, 4)  # 24595658764946068820
comper(64, 4)  # 453709822561251284617832809909024281940
```

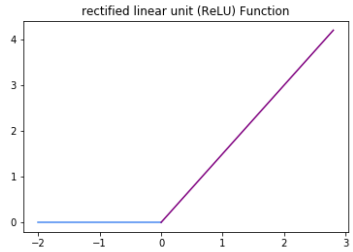
[12] *Python* 的主要对象单元有:*int, float, str, tuple, list, range, dict.* 其中后四个是具有一定结构的数据单元，很多更高层级的程式和扩展都是依托在这四个结构数据单元上建立的。比如我们使用的 *Numpy*，很多指令是依托在 *list* 上，即 *[ ]*, 所以你在使用 *numpy* 的时候，数据的输入都要加上中括号。有关 *Python* 的语言结构和特性我们会再具体的应用时，酌情进行讲解。

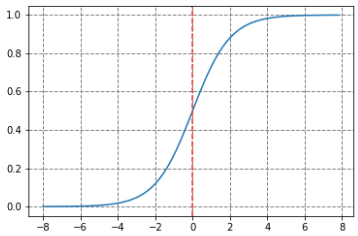
```
# HOPE, now you understand why we need quantum computer
# 希望现在你能够理解为什么我们需要量子计算器了
```

C1-Q5 的 Python 作图。[13]

```
# Chapter 3 Pre-lecture Exercise
# 第三章课前练习
# @ Michael

# Import essential packages
# 导入必要扩展

import numpy as np
import matplotlib.pyplot as plt

# Play with colours in Python
# 简单的颜色绘图
# We will use matplotlib to plot a very simple function
# with varied colours
# 我们将会使用 matplotlib 绘制简单的方程图形并且调试不同颜色

plt.plot([2], [3], 'bo')
plt.plot([4], [4], 'ro')
plt.plot([5], [2], 'go')
plt.axis([0, 6, 0, 7])  # change the range of axis
plt.grid(color='gray', linestyle='--', linewidth=1)  # add grid
plt.title('My first plot')


# plot ReLU function
x1 = np.arange(-2, 0.2, 0.2)
x2 = np.arange(0, 3, 0.2)
plt.plot(x1, 0*x1, color=(70/255, 136/255, 241/255))
# RGB(70, 136, 241)
plt.plot(x2, 1.5*x2, 'purple')
plt.title('rectified linear unit (ReLU) Function')


# plot sigmoid function
x3 = np.arange(-8, 8, 0.2)
y3 = np.exp(x3)/(1+np.exp(x3))
plt.plot(x3, y3)
plt.grid(color='gray', linestyle='--', linewidth=1)  # add grid
plt.axvline(x=0, color=(243/255, 66/255, 53/255), linestyle='--')
# add vertial line
```

[13] 就像我们再 *C1-Q5* 中手动作图一样，*Python* 在作图时也需要你给出 X—Y 上的点。计算机作图和数据可视化已经可以成为单独的一门学科，比较数据可视化比较常用的编程语言是 *Javascript*，比如 *D3.js* 和 *Chart.js*。你在网络上看到的很多的数据动画都是使用以上两种语言制作的。此外,*R* 语言的数据可视化也比较流行，*Python* 正在奋起直追,比如最新的 *Python* 作图拓展 *Seaborn*。





3.2 课堂讲义 C2

3.3 课后练习 C3