机器学习算法

感知机

我真的不懂忧郁

Delft University of Technology

TU Delft

目录

Nomenclature **iii**

1 感知机 (Perceptron) **1**
1.1 超平面 . . . 1
1.2 模型 . . . 2
1.3 数据集的线性可分性 . . . 2
1.4 损失函数 . . . 2
1.5 感知机学习算法 . . . 3
1.6 感知机收敛定理 . . . 4

2 感知机代码仿真 **6**
2.1 算法描述 . . . 6
2.2 实战 . . . 7

3 训练感知机实现逻辑运算 **10**
3.1 感知机的逻辑运算 . . . 10
3.2 异或问题 . . . 11
3.3 多层感知机 . . . 11

4 扩展阅读——通用近似定理 **12**

5 参考阅读 **14**

A Source Code Example **15**

B Latex Tikz Test Code **16**

C Task Division Example **18**

Nomenclature

If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

Abbreviations

Abbreviation	Definition
ISA	International Standard Atmosphere
...	

Symbols

Symbol	Definition	Unit
V	Velocity	[m/s]
...		
ρ	Density	[kg/m^3]
...		

Chapter 1

感知机 (Perceptron)

1.1. 超平面

假设 $\mathbb{R}^n$ 空间中的一个平面 S，S 上的一个法向量为

$$\vec{n} = (\omega_1, \omega_2, \cdots, \omega_n) \tag{1.1}$$

固定平面 S 的一个点 $P_0 = (x_0, x_1, \cdots, x_n)$，对于 S 上任意一个点 $P = (x'_1, x'_2, \cdots, x'_n)$，构成一个向量

$$\vec{x} = \vec{PP_0} = (x'_1 - x_1, x'_2 - x_2, \cdots, x'_n - x_n) \tag{1.2}$$

对于平面上任意一个向量，都与法向量垂直，因此平面方程表示为法向量和平面上任意一个向量的内积

$$\vec{n} \cdot \vec{x} = 0 \tag{1.3}$$

三维空间的平面称为平面，如果是任意的维度，广义的平面称为超平面。

实例

对于平面上的点，因为都与超平面垂直，因此如果实例点落入超平面中，$\vec{n} \cdot \vec{x} = 0$。对于位于超平面上方的点，$PP_0$ 和法向量 n 之间的夹角

$$\frac{PP_0 \cdot n}{\|PP_0\|\|n\|} \leqslant 90° \tag{1.4}$$

因此在超平面上方的点都满足

$$PP_0 \cdot n \geqslant 0 \tag{1.5}$$

反之，超平面下方的点满足

$$PP_0 \cdot n \leqslant 0 \tag{1.6}$$

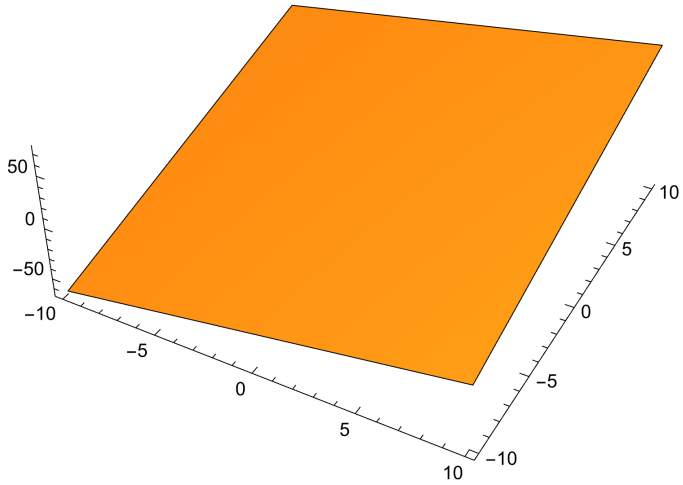


图 1.1: 超平面

1.2. 模型

definition 1.2.1: *(感知机)* 假设输入空间（特征空间）$\mathcal{X} \subset \mathbb{R}^n$，输出空间是 $\dagger = \{-1, +1\}$，输入 $x \in \mathcal{X}$ 表示实例的特征向量，对应于输入空间（特征空间）的点；输出 $y \in \dagger$ 表示实例的类别，由输入空间到输出空间的如下符号函数

$$f(x) = sign(\omega \cdot x + b) \tag{1.7}$$

称为感知机。其中，ω 和 b 为感知机模型参数。

1.3. 数据集的线性可分性

假设一个数据集

$$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\} \tag{1.8}$$

如果存在一个超平面 $\omega \cdot x + b = 0$，使得数据集的正实例点和负实例点完全正确地划分到超平面的两侧处，即对所有的 $y_i = +1$ 的实例 i，有

$$\omega \cdot x_i + b > 0 \tag{1.9}$$

对所有的 $y_i = -1$ 的实例 i，

$$\omega \cdot x_i + b < 0 \tag{1.10}$$

则 T 被称为线性可分的数据集。

1.4. 损失函数

实例点误分类的判别关系

根据输入空间到输出空间的映射方式，误分类的实例点应该满足下面的关系。

$$-y_i(\omega \cdot x_i + b) > 0 \tag{1.11}$$

感知机的损失函数

损失函数一个自然选择是误分类点的总数，但是这样损失函数就不是参数 ω 和 b。但是可以考虑：如果所有误分类点距离超平面的距离为 0，则这是一次完美的分割。

输入空间 R^n 中任意一点 x 到超平面 S 到距离为

$$\frac{1}{\|\omega\|}|\omega \cdot x + b| \tag{1.12}$$

对于误分类的数据来说，距离为

$$\frac{1}{\|\omega\|} y_i(\omega \cdot x + b) \tag{1.13}$$

[1]

所有误分类点到超平面 S 到总距离为

$$L = -\frac{1}{\|\omega\|} \sum_{x_i \in M}^{n} y_i(\omega \cdot x_i + b) \tag{1.14}$$

1.5. 感知机学习算法

给定一个训练数据集

$$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\} \tag{1.15}$$

其中 $x_i \in \mathcal{X} = R^n$，$y_i \in \mathcal{Y} = \{-1, 1\}$，$i = 1, 2, \cdots, N$，求参数 ω、b 使得损失函数最小化

$$\min_{\omega,b} L(\omega, b) = -\sum_{x_i \in M} y_i(\omega \cdot x_i + b) \tag{1.16}$$

其中 M 是误分类点集合。

原始形式

输入：训练数据集 T；设定学习率 η $(0 < \eta \leqslant 1)$；
输出：$\omega, b; f(x) = sign(\omega \cdot x + b)$；

1. 选定初值 ω_0, b_0；
2. 在训练集中选取数据 (x_i, y_i)；
3. 如果 $y_i(\omega \cdot x_i + b) \leqslant 0$，更新权重

$$\begin{aligned} \omega &\leftarrow \omega + \eta y_i x_i \\ b &\leftarrow b + \eta y_i \end{aligned} \tag{1.17}$$

4. 跳转至 2，直到没有误分类点。

[1] 因为 y_i 只能去 +1 和-1，因此可以把绝对值符号去掉

对偶形式

输入：训练数据集 T；设定学习率 $\eta\ (0<\eta\leqslant 1)$；
输出：$\alpha, b; f(x)=sign(\sum_{j=1}^{N}\alpha_j y_j x_j\cdot x+b)$，其中 $\alpha=(\alpha_1,\alpha_2,\cdots,\alpha_N)^T$；

1. $\alpha\leftarrow 0$，$b\leftarrow 0$；
2. 训练集上选取数据 (x_i,y_i)；
3. 如果 $y_i(\sum_{j=1}^{N}\alpha_j y_j x_j\cdot x_i+b)\leqslant 0$

$$\begin{aligned}\alpha_i &\leftarrow \alpha_i+\eta\\ b &\leftarrow b+\eta y_i\end{aligned}\tag{1.18}$$

4. 跳转到 2 直到没有误分类点数据。

对偶形式的基本想法是，将 ω 和 b 表示为实例 x_i 和 y_i 的线性组合的形式，通过求解其系数而求得 ω 和 b。不失一般性，在原始形式算法中假设初始值均为 0，对误分类点 (x_i,y_i) 通过

$$\begin{aligned}\omega &\leftarrow \omega+\eta y_i x_i\\ b &\leftarrow b+\eta y_i\end{aligned}\tag{1.19}$$

逐步修改 ω 和 b，修改 n 次的 ω 和 b 增量为 $\alpha_i y_i x_i$ 和 $\alpha_i y_i$。这里 $\alpha_i=n_i\eta$。这样，从学习过程不难看出，最后学习到的 ω, b 可以分别表示为

$$\omega=\sum_{i=1}^{N}\alpha_i y_i x_i\tag{1.20}$$

$$b=\sum_{i=1}^{N}\alpha_i y_i\tag{1.21}$$

这里 $\alpha_1\geqslant 1$，$i=1,2,\cdots,N$，当 $\eta=1$ 时，表示第 i 个实例点由于误分类而进行更新的次数。实例点更新次数越多，意味着它距离超平面越近，也就越难正确分类。这样的实例对学习结果影响最大。

1.6. 感知机收敛定理

theorem 1.6.1: *(Novikoff)* 设训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$ 是线性可分的数据集，其中 $x_i\in\mathcal{X}=R^n$，$y_i\in\mathcal{Y}=-1,+1$，$i=1,2,\cdots,N$，则

1. 存在满足条件 $\|\hat{\omega}_{opt}=1\|$ 的超平面 $\hat{\omega}_{opt}\cdot\hat{x}=\omega_{opt}\cdot x+b_{opt}=0$ 将训练数据集完全正确分开；且存在 $\gamma>0$，对于所有 $i=1,2,\cdots,N$

$$y_i(\hat{\omega}_{opt}\cdot\hat{x})=y_i(\omega_{opt}\cdot x_i+b_{opt})\geqslant\gamma\tag{1.22}$$

2. 令 $R=\max\limits_{1\leqslant i\leqslant N}\|\hat{x}_i\|$，则感知机算法在训练数据集上的误分类次数 k 满足不等式

$$k\leqslant(\frac{R}{\gamma})^2\tag{1.23}$$

proof. 首先证明第一点：

由于训练集数据是线性可分的，按照定义，存在超平面可以将数据集完全正确分开，取此超平面为

$$\hat{\omega}_{opt} \cdot \hat{x} = \omega_{opt} \cdot + b_{opt} = 0 \tag{1.24}$$

由于等式右边等于 0，所以总可以取 $\|\hat{\omega}\| = 1$。对于有限的 $i = 1, 2, \cdots, N$，正确分类下有

$$y_i(\hat{\omega}_{opt} \cdot \hat{x}_i) = y_i(\omega_{opt} \cdot x_i + b_{opt}) > 0 \tag{1.25}$$

所以 0 是序列 $\{y_i(\hat{\omega}_{opt} \cdot \hat{x}_i)\}$ 的一个下界，假设序列的下确界

$$\gamma = \inf_i \{y_i(\omega_o pt \cdot x_i + b_{opt})\} \tag{1.26}$$

使得

$$y_i(\hat{\omega}_{opt} \cdot \hat{x}_i) = y_i(\omega_{opt} \cdot x_i + b_{opt}) \geqslant \gamma \geqslant 0 \tag{1.27}$$

下面证明第二点：

训练点为 x_i，标签为 y_i。假设 $\hat{\omega}_{opt} = 0$ 开始，如果实例被误分类，则更新权重。令 $\hat{\omega}_{k-1}$ 是第 k 个误分类实例之前的扩充权重向量，即

$$\hat{\omega}_{k-1} = (\omega_{k-1}^T, b_{k-1})^T \tag{1.28}$$

如果第 k 个实例点是误分类点，则

$$y_i(\hat{\omega}_{k-1} \cdot \hat{x}_i) = y_i(\omega_{k-1} \cdot x_i + b_{k-1}) \leqslant 0 \tag{1.29}$$

在误分类的时候更新权重

$$\begin{aligned} \omega_k &\leftarrow \omega_{k-1} + \eta y_i x_i \\ b_k &\leftarrow b_{k-1} + \eta y_i \end{aligned} \tag{1.30}$$

即

$$\hat{\omega}_k = \hat{\omega}_{k-1} + \eta y_i \hat{x}_i \tag{1.31}$$

利用两个不等式，这两个不等式将在后续的引理中证明。

$$\hat{\omega}_k \cdot \hat{\omega}_{opt} \geqslant k\eta\gamma \tag{1.32}$$

$$\|\hat{\omega}_k\|^2 \leqslant k\eta^2 R^2 \tag{1.33}$$

代入方程 (1.31)，得到

$$\begin{aligned} &k\eta\gamma \leqslant \hat{\omega}_k \cdot \hat{\omega}_{opt} \leqslant \|\hat{\omega}_k\| \|\hat{\omega}_{opt}\| \leqslant \sqrt{k}\eta R \\ &k^2\gamma^2 \leqslant kR^2 \end{aligned} \tag{1.34}$$

于是

$$k \leqslant (\frac{R}{\gamma})^2 \tag{1.35}$$

Chapter 2

感知机代码仿真

2.1. 算法描述

代码实现思路为：初始化权重，如果实例点正确分类，则检查下一个实例点，指标指向下一个实例点，如果实例点被误分类，则更新权重，指标回到第一个检查的实例点重新开始检查，直到找到权重使得从第一个到最后一个实例点都被正确分类。

```
import numpy as np
import matplotlib.pyplot as plt

# 定义感知机模型类
class Perceptorn:
    def __init__(self):
        self.count = 1
        self.w = None
        self.b = 0
        self.l_late = 1

    def fig(self):
        # 模型方程
        x_plot = np.linspace(2, 7,10)
        y_plot = -(perceptorn.w[0]*x_plot+perceptorn.b)/perceptorn.w[1]
        # 图示
        plt.plot(x_plot,y_plot)
        plt.scatter(x_trian[0:3,0],x_trian[0:3,1],marker='+', color='orange',label='postive')
        plt.scatter(x_trian[3:6,0],x_trian[3:6,1],marker='+', color='green',label='negtive')

        #设置图例
        plt.legend(loc=1)

        #显示
        plt.show()

    # 算法主函数
    def fit(self,x_trian,y_trian):
        cnt=0
```

```
30          self.w=np.zeros(x_trian.shape[1])#shape 方法通常用于获取数组或矩阵的形状（维度），主要用于
                NumPy 库中的数组对象。
31          i = 0
32
33          while i < x_trian.shape[0]:
34              x = x_trian[i]
35              y = y_trian[i]
36              # 判断其所有点是否都没有误分类，如有更新w,b,重新判断
37              if y*(np.dot(self.w,x)+self.b)<=0:
38                  self.w = self.w + self.l_late*np.dot(x,y)
39                  self.b = self.b+self.l_late*y
40                  i = 0
41              else:
42                  i += 1
43
44              cnt=cnt+1
45
46          print("trainning finish ! trainning cost times : {}".format(cnt))
```

利用感知机模型，对下列的测试数据进行训练

```
1 # 训练集
2 x_trian = np.array([[3,3],[3,5],[4,7],[4,0],[2,1],[6,2]])
3 y_trian = np.array([1,1,1,-1,-1,-1])
4 print(x_trian.size,y_trian.size)
```

调用 Perceptron 类对数据进行训练，得到结果如下

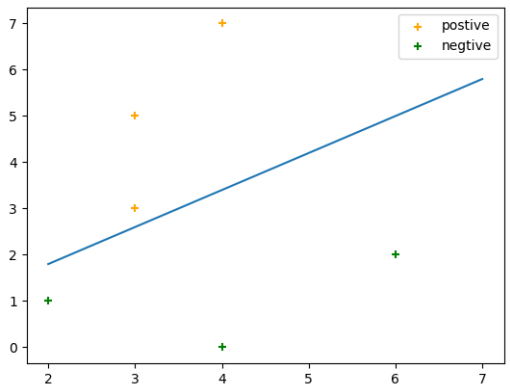


图 2.1: 训练成果

算法的训练次数为 21 次，权重向量为 $[-4.5.]$，偏置为 -1。

2.2. 实战

下面的例子是 *sklearn* 里面的鸢尾花的例子。首先导入需要的包和数据集。

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.datasets import load_iris
4 from sklearn.linear_model import Perceptron
5 import matplotlib.pyplot as plt
```

加载数据集，并且把数据集转换为 Dataframe 类型。

```
# 处理数据
iris = load_iris()
data = pd.DataFrame(iris.data,columns=iris.feature_names)
```

DataFrame 是 pandas 中最重要的数据结构之一，类似于 Excel 表格或 SQL 表。DataFrame 可以包含多种类型的数据，包括整数、浮点数、字符串等，并且可以轻松地进行数据操作和分析。

鸢尾花的数据表格如下所示

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

图 2.2: 鸢尾花数据集

其中分为花萼的长宽和花瓣的长宽，接下来加载标签值。

```
data['label'] = iris.target #标签值
data.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
print(data.head(4))
```

标签值如下

	sepal length	sepal width	petal length	petal width	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0

图 2.3: 鸢尾花数据集

我们用花萼的数据来训练模型，可视化数据集图像如下

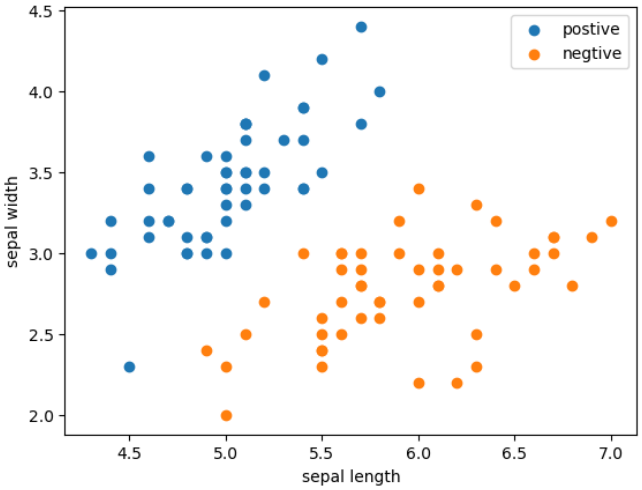


图 2.4: 鸢尾花数据集的二维分布

可以直观地看出，数据集明显是线性可分的。我们例化 sklearn 中的感知机模型，代码如下

```
x_train = np.array(data.iloc[:100,[0,1]])
y_train = np.array(data.iloc[:100,-1])

perceptron = Perceptron()
perceptron.fit(x_train,y_train)
print('w:',perceptron.coef_,'\n','b:',perceptron.intercept_,'\n','迭代次数:',perceptron.n_iter_)

# 评判模型
res = perceptron.score(x_train, y_train)
print('成功率:{:.0%}'.format(res))
```

拟合次数为 8 次，得到的拟合超平面如下

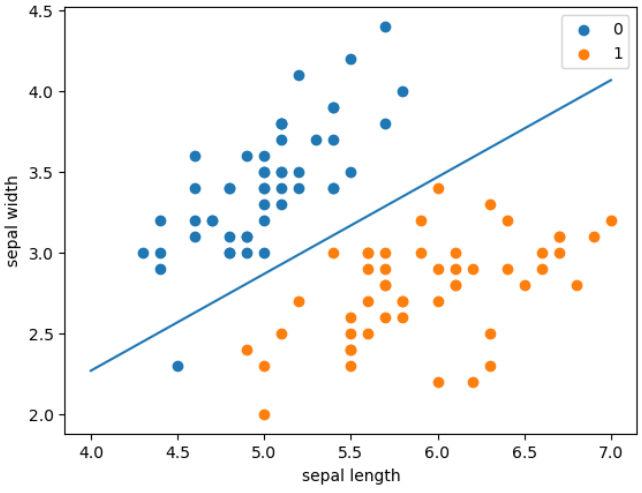


图 2.5: 鸢尾花数据集的二维分布

可见拟合结果中有一个被误分类，但是总体上是成功分类了。

Chapter 3

训练感知机实现逻辑运算

3.1. 感知机的逻辑运算

简单的逻辑运算也是线性可分问题，感知机模型处理逻辑运算的超平面如下图所示

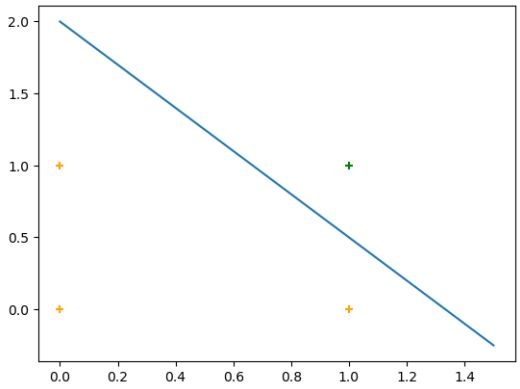


图 3.1: 与运算

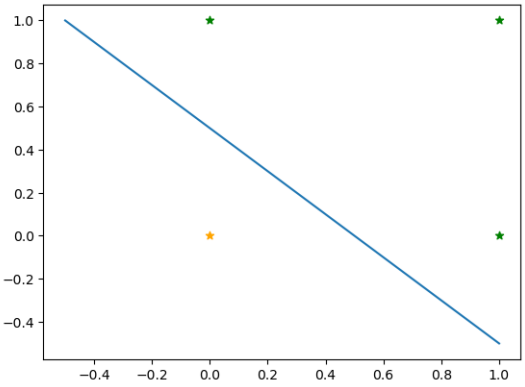
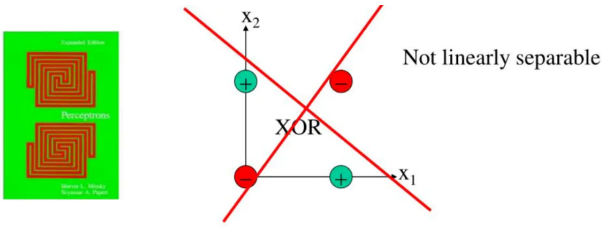


图 3.2: 或运算

3.2. 异或问题

单层感知机的一个问题是不能实现异或运算。



Minsky & Papert (1969)
Perceptrons can only represent linearly separable functions.

图 3.3: 异或问题

如下图所示，异或问题不是一个线性可分的问题，因此单层感知机在实现逻辑运算上存在缺陷。

3.3. 多层感知机

多层感知器（英语：Multilayer Perceptron，缩写：MLP）是一种前向结构的人工神经网络，映射一组输入向量到一组输出向量。MLP 可以被看作是一个有向图，由多个的节点层所组成，每一层都全连接到下一层。除了输入节点，每个节点都是一个带有非线性激活函数的神经元（或称处理单元）。一种被称为反向传播算法的监督学习方法常被用来训练 MLP。

多层感知器遵循人类神经系统原理，学习并进行数据预测。它首先学习，然后使用权重存储数据，并使用算法来调整权重并减少训练过程中的偏差，即实际值和预测值之间的误差。主要优势在于其快速解决复杂问题的能力。多层感知的基本结构由三层组成：第一输入层，中间隐藏层和最后输出层，输入元素和权重的乘积被馈给具有神经元偏差的求和结点，主要优势在于其快速解决复杂问题的能力。MLP 是感知器的推广，克服了感知器不能对线性不可分数据进行识别的弱点。

Chapter 4

扩展阅读——通用近似定理

在人工神经网络的数学理论中，通用近似定理（或称万能近似定理）指出人工神经网络近似任意函数的能力。通常此定理所指的神经网络为前馈神经网络，并且被近似的目标函数通常为输入输出都在欧几里得空间的连续函数。但亦有研究将此定理扩展至其他类型的神经网络，如卷积神经网络、放射状基底函数网络、或其他特殊神经网络。

此定理意味着神经网络可以用来近似任意的复杂函数，并且可以达到任意近似精准度。但它并没有说明要如何选择神经网络参数（权重、神经元数量、神经层层数等等）来达到想近似的目标函数。

希尔伯特第十三问题

德国数学家希尔伯特希望数学界能够证明：

$$f^7 + xf^3 + yf^2 + zf + 1 = 0 \tag{4.1}$$

的七个解若表示成系数为 x, y, z 的函数，则此函数无法简化成两个变数的函数。

1957 年，苏联数学家安德雷 · 柯尔莫哥洛夫（Андре́й Никола́евич Колмого́ров）的学生、当时 19 岁的弗拉基米尔 · 阿诺尔德（Влади́мир И́горевич Арно́льд）解决了这个问题。柯尔莫哥洛夫证明每个有多个变元的函数可用有限个三变元函数构作。阿诺尔德按这个结果研究，证明两个变元已足够。之后阿诺尔德和日本数学家志村五郎发表了一篇论文（Superposition of algebraic functions (1976), in Mathematical Developments Arising From Hilbert's Problems）。这些结果后来被进一步发展，推导出人工神经网络中的通用近似定理，指人工神经网络能近似任意连续函数。

Kolmogorov–Arnold 表示定理

在实分析和逼近理论中，Kolmogorov-Arnold 表示定理（Kolmogorov–Arn old representation theorem，或叠加定理（superposition theorem））指出，每个多元连续函数都可以表示为有限数量的单变量连续函数的两层嵌套叠加，这两个层的函数分别称为内部函数和外部函数。它解决了希尔伯

特第十三问题的一个更受约束但更一般的形式。[1]

[1]米田引理、近似理论和迦瓦罗理论

Chapter 5

参考阅读

1. 《希尔伯特第 13 问题，Kolmogorov–Arnold representation theorem 和通用近似定理》（Universal approximation theorem）https://blog.csdn.net/qq_32515081/article/details/129569496
2. 《Understanding the Universal Approximation Theorem 》https://towardsai.net/p/deep-learning/understanding-the-universal-approximation-theorem
3. 泛函分析：Stone–Weierstrass theorem
4. https://www.jiqizhixin.com/articles/2021-09-07-6

Chapter A

Source Code Example

Adding source code to your report/thesis is supported with the package `listings`. *An example can be found below. Files can be added using* `\lstinputlisting[language=<language>]{<filename>}`.

```
"""
ISA Calculator: import the function, specify the height and it will return a
list in the following format: [Temperature,Density,Pressure,Speed of Sound].
Note that there is no check to see if the maximum altitude is reached.
"""

import math
g0 = 9.80665
R = 287.0
layer1 = [0, 288.15, 101325.0]
alt = [0,11000,20000,32000,47000,51000,71000,86000]
a = [-.0065,0,.0010,.0028,0,-.0028,-.0020]

def atmosphere(h):
    for i in range(0,len(alt)-1):
        if h >= alt[i]:
            layer0 = layer1[:]
            layer1[0] = min(h,alt[i+1])
            if a[i] != 0:
                layer1[1] = layer0[1] + a[i]*(layer1[0]-layer0[0])
                layer1[2] = layer0[2] * (layer1[1]/layer0[1])**(-g0/(a[i]*R))
            else:
                layer1[2] = layer0[2]*math.exp((-g0/(R*layer1[1]))*(layer1[0]-layer0[0]))
    return [layer1[1],layer1[2]/(R*layer1[1]),layer1[2],math.sqrt(1.4*R*layer1[1])]
```

Chapter B

Latex Tikz Test Code

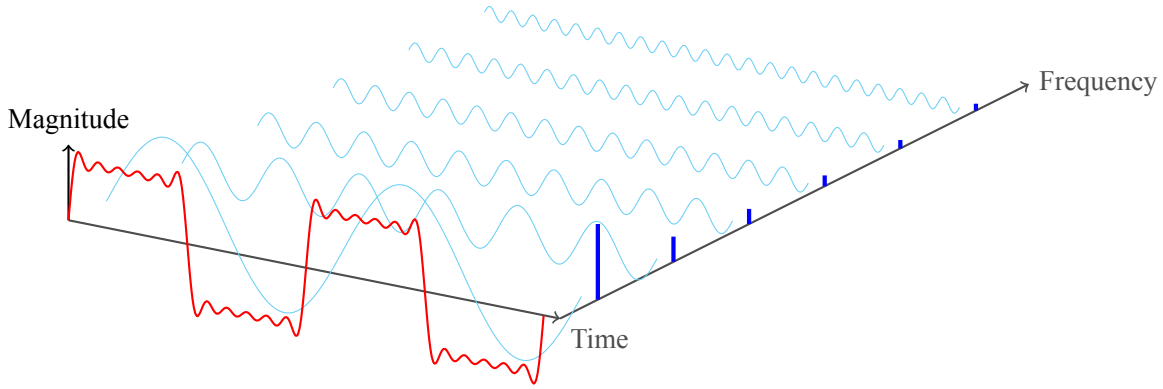


图 B.1: 测试图 1

```
\begin{figure}[H]
    \begin{tikzpicture}[x={(1cm,0.5cm)},z={(0cm,0.5cm)},y={(1cm,-0.2cm)}]%指定坐标系的方向与缩放
        \draw[->,thick,black!70] (0,6.5,0) -- (6.2,6.5,0) node[right] {Frequency}; % 频率轴
        \draw[->,thick,black!70] (0,0,0) -- (0,6.5,0) node[below right] {Time}; % 时间轴
        \draw[->,thick] (0,0,0) -- (0,0,2) node[above] {Magnitude};

        \foreach \n in {0.5,1.5,...,5.5}{
        \draw [cyan!50, domain=0:2*pi,samples=200,smooth]
         plot (\n,\x, {sin(4*\n*\x r)/\n });
        \draw[blue, ultra thick] (\n,6.5,0) -- (\n,6.5,1/\n);
        } % 频率逐渐增大振幅逐渐变小的正弦函数

        \draw [red, thick, domain=0:2*pi,samples=200,smooth]
        plot (0,\x, {sin(4*0.5*\x r)/0.5 + sin(4*1.5*\x r)/1.5 + sin(4*2.5*\x r)/2.5
         + sin(4*3.5*\x r)/3.5 + sin(4*4.5*\x r)/4.5 + sin(4*5.5*\x r)/5.5} );
        % 最后是手动加起来得到矩形波的逼近
        \end{tikzpicture}
        \caption{测试图1}
```

```
        \end{figure}
```

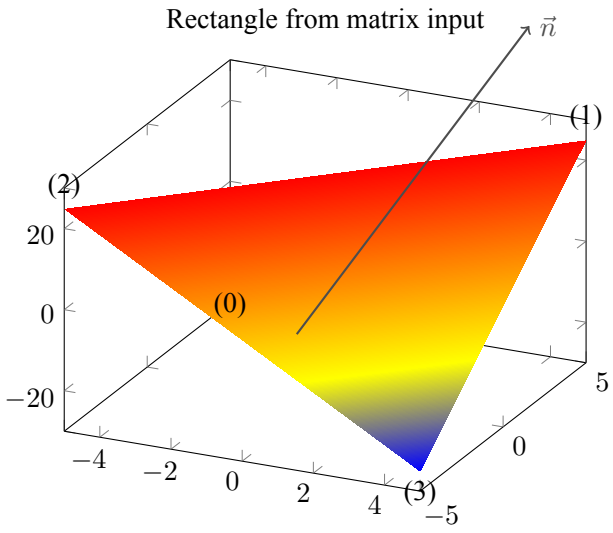


图 B.2: 三维绘图

```

    \begin{tikzpicture}
        \begin{axis}[nodes near coords={(\coordindex)},
            title=Rectangle from matrix input]
        % note that surf implies 'patch type=rectangle'
        \addplot3[surf,shader=interp,samples=2,
            patch type=rectangle]
            {x*y};
        \end{axis}
    \end{tikzpicture}
```

Chapter C

Task Division Example

If a task division is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

表 C.1: Distribution of the workload

	Task	Student Name(s)
	Summary	
Chapter 1	Introduction	
Chapter 2		
Chapter 3		
Chapter *		
Chapter *	Conclusion	
	Editors	
	CAD and Figures	
	Document Design and Layout	