姓名：杜兴豪
学号：201300096

一. (20 points) **神经网络基础**

给定训练集 $D=\{(\boldsymbol{x}_1,\boldsymbol{y}_1),(\boldsymbol{x}_2,\boldsymbol{y}_2),...,(\boldsymbol{x}_m,\boldsymbol{y}_m)\}$. 其中 $\boldsymbol{x}_i\in\mathbb{R}^d$, $\boldsymbol{y}_i\in\mathbb{R}^l$ 表示输入示例由 d 个属性描述, 输出 l 维实值向量. 图 1给出了一个有 d 个输入神经元、l 个输出神经元、q 个隐层神经元的多层神经网络, 其中输出层第 j 个神经元的阈值用 θ_j 表示, 隐层第 h 个神经元的阈值用 γ_h 表示. 输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih}, 隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj}. 记隐层第 h 个神经元接收到的输入为 $\alpha_h=\sum_{i=1}^d v_{ih}x_i$, 输出层第 j 个神经元接收到的输入为 $\beta_j=\sum_{h=1}^q w_{hj}b_h$, 其中 b_h 为隐层第 h 个神经元的输出.

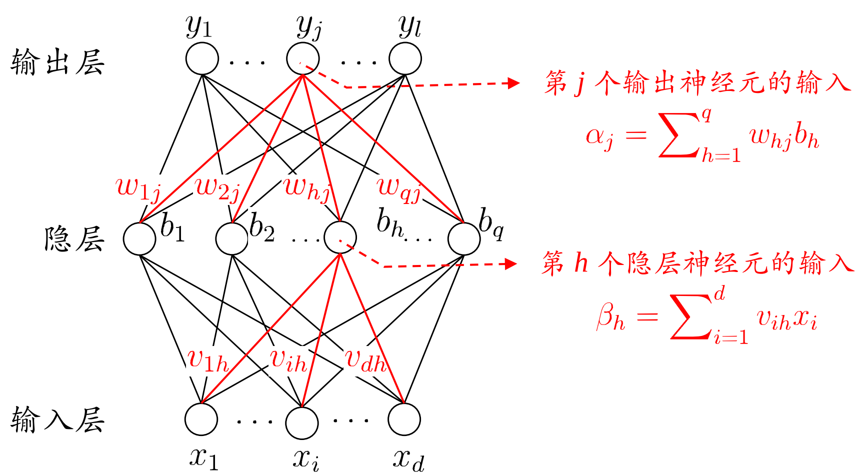

Figure 1: 多层神经网络（教材图 5.7）

不同任务中神经网络的输出层往往使用不同的激活函数和损失函数, 本题介绍几种常见的激活和损失函数, 并对其梯度进行推导.

1. 在二分类问题中（$l=1$）, 标记 $y\in\{0,1\}$, 一般使用 Sigmoid 函数作为激活函数, 使输出值在 $[0,1]$ 范围内, 使模型预测结果可直接作为概率输出. Sigmoid 函数的输出一般配合二元交叉熵 (Binary Cross-Entropy) 损失函数使用, 对于一个训练样本 $(\boldsymbol{x},y)$ 有

$$\ell(y,\hat{y}_1)=-\left[y\log(\hat{y}_1)+(1-y)\log(1-\hat{y}_1)\right] \tag{1}$$

记 $\hat{y}_1$ 为模型对样本属于正类的预测结果, 请计算 $\frac{\partial\ell(y,\hat{y}_1)}{\partial\beta_1}$,

2. 当 $l > 1$, 网络的预测结果为 $\hat{\boldsymbol{y}} \in \mathbb{R}^l$, 其中 $\hat{y}_i$ 表示输入被预测为第 i 类的概率. 对于第 i 类的样本, 其标记 $\boldsymbol{y} \in \{0,1\}^l$, 有 $y_i = 1$, $y_j = 0, j \neq i$. 对于一个训练样本 $(\boldsymbol{x}, \boldsymbol{y})$, 交叉熵损失函数 $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$ 的定义如下

$$\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{l} y_j \log \hat{y}_j \tag{2}$$

在多分类问题中, 一般使用 Softmax 层作为输出, Softmax 层的计算公式如下

$$\hat{y}_j = \frac{e^{\beta_j}}{\sum_{k=1}^{l} e^{\boldsymbol{\beta}_k}} \tag{3}$$

易见 Softmax 函数输出的 $\hat{\boldsymbol{y}}$ 符合 $\sum_{j=1}^{l} \hat{y}_j = 1$, 所以可以直接作为每个类别的概率. Softmax 输出一般配合交叉熵 (Cross Entropy) 损失函数使用, 请计算 $\frac{\partial \ell(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \beta_j}$,

3. 分析在二分类中使用 Softmax 和 Sigmoid 的联系与区别.
4. KL 散度 (Kullback-Leibler divergence) 定义了两个分布之间的距离, 对于两个离散分布 $Q(x)$ 和 $P(x)$, 其定义为

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{4}$$

其中 $\mathcal{X}$ 为 x 的取值空间. 试分析交叉熵损失函数和 KL 散度的关系.

解:

1. 由于采用 Sigmoid 函数作为激活函数，故

$$f(x) = \frac{1}{1 + e^{-x}}$$

因此有: 当 $l = 1$ 时，得到输入和输出的关系为

$$\hat{y}_1 = f(\beta_1 - \theta_1)$$

则有

$$\frac{\partial \hat{y}_1}{\partial \beta_1} = \frac{e^{-(\beta_1 - \theta_1)}}{(1 + e^{-(\beta_1 - \theta_1)})^2}$$

因此可计算

$$\begin{aligned}\frac{\partial \ell(y,\hat{y}_1)}{\partial \beta_1} &= \frac{\partial \ell(y,\hat{y}_1)}{\partial \hat{y}_1}\cdot\frac{\partial \hat{y}_1}{\partial \beta_1} = \left(\frac{1-y}{1-\hat{y}_1}-\frac{y}{\hat{y}_1}\right)\cdot\frac{\partial \hat{y}_1}{\partial \beta_1}\\ &= \left(\frac{(1-y)(1+e^{-(\beta_1-\theta_1)})}{e^{-(\beta_1-\theta_1)}} - y(1+e^{-(\beta_1-\theta_1)})\right)\cdot\frac{e^{-(\beta_1-\theta_1)}}{(1+e^{-(\beta_1-\theta_1)})^2}\\ &= \frac{1}{1+e^{-(\beta_1-\theta_1)}} - y\end{aligned}$$

2. 易知

$$\begin{aligned}&\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}\\ &\frac{\partial \hat{y}_i}{\partial \beta_j} = -\frac{e^{\beta_i}}{\left(\sum_{k=1}^{l} e^{\beta_k}\right)^2} \quad ,i\neq j\\ &\frac{\partial \hat{y}_j}{\partial \beta_j} = \frac{\sum\limits_{k=1,k\neq j}^{l} e^{\beta_k+\beta_j}}{\left(\sum_{k=1}^{l} e^{\beta_k}\right)^2}\end{aligned}$$

因此有

$$\begin{aligned}\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \beta_j} &= \sum_{i=1}^{l}\frac{\partial \ell(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{y}_i}\cdot\frac{\partial \hat{y}_i}{\partial \beta_j}\\ &= -\frac{y_j}{\hat{y}_j}\cdot\frac{\sum_{k=1,k\neq j}^{l} e^{\beta_k+\beta_j}}{\left(\sum_{k=1}^{l} e^{\beta_k}\right)^2} + \sum_{i=1,i\neq j}^{l}\frac{y_i}{\hat{y}_i}\cdot\frac{e^{\beta_i}}{\left(\sum_{k=1}^{l} e^{\beta_k}\right)^2}\\ &= \frac{\sum\limits_{k=1,k\neq j}^{l} y_k e^{\beta_k} - y_j e^{\beta_j}\sum\limits_{k=1,k\neq j}^{l} e^{\beta_k}}{\sum_{k=1}^{l} e^{\beta_k}}\end{aligned}$$

3. 联系：Softmax 函数满足 $\sum_{i=1}^{l}\hat{y}_i = 1$，因此可直接将输出看作是样本被判定为每一类别的概率。而 Sigmoid 函数由于分布在 (0,1) 范围之内，也可以看作是概率。
区别：Sigmoid 函数只能判断样本属于给定类的概率，在二分类

问题中，属于隐式地给出了属于另一类的概率（不属于给定类，则属于另一类）。而 Softmax 则显式地指出了样本属于各类的概率。

4.
$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \\ &= \sum_{x\in\mathcal{X}} P(x)\left(\log P(X) - \log Q(x)\right) \\ &= \sum_{x\in\mathcal{X}} P(x)\log P(x) - \sum_{x\in\mathcal{X}} P(x)\log Q(x) \end{aligned}$$

在分类问题中，下标范围即为取值空间，定义

$$\begin{aligned} &\mathcal{X} = \{\, i \mid 1 \le i \le l, i \in N\} \\ &P(x) = y_i \\ &Q(x) = \hat{y}_i \end{aligned}$$

则原来的 KL 散度改写为

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{i=1}^{l} y_i \log y_i - \sum_{i=1}^{l} y_i \log \hat{y}_i$$

由于 y_i 的分布为固定的，因此上式的前半部分为一个常数，记为 $f(\boldsymbol{y})$，我们有

$$D_{\mathrm{KL}}(P \parallel Q) = f(\boldsymbol{y}) + \ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

上式即为交叉熵损失函数和 KL 散度的关系

二. (20 points) **运算的向量化**

在编程实践中, 一般需要将运算写成向量或者矩阵运算的形式, 这叫做运算的向量化 (vectorization). 向量化可以充分利用计算机体系结构对矩阵运算的支持加速计算, 大部分数学运算库例如`numpy`也对矩阵计算有专门的优化. 另一方面, 如果一个运算可以写成向量计算的形式, 会更容易写出其导数形式并进行优化. 本题中举两个简单的例子

1. 给定示例矩阵 $\boldsymbol{X} \in \mathbb{R}^{m\times d}$, 表示 m 个示例（向量）, 每个示例有 d 维,

计算 m 个示例两两之间的距离矩阵 $\boldsymbol{D} \in \mathbb{R}^{m\times m}$, 两个向量之间的欧式距离定义为 $\|\boldsymbol{x}-\boldsymbol{y}\|_2 = \sqrt{\sum_{i=1}^{d}(x_i-y_i)^2}$. 求距离矩阵可以通过循环的方式, 即plain_distance_function中实现的方法;

```python
import numpy as np

def plain_distance_function(X):
    # 直观的距离计算实现方法
    # 首先初始化一个空的距离矩阵D
    D = np.zeros((X.shape[0], X.shape[0]))
    # 循环遍历每一个样本对
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            # 计算样本i和样本j的距离
            D[i, j] = np.sqrt(np.sum((X[i] - X[j])**2))
    return D
```

2. 输入一个矩阵 $\boldsymbol{X} \in \mathbb{R}^{m\times d}$, 表示 m 个向量, 每个向量有 d 维, 要求对输入矩阵的行按照一个给定的排列 $\boldsymbol{p} = \{p_1, p_2, ..., p_m\}$ 进行重新排列. 即输出一个新的矩阵 $\boldsymbol{X}'$, 其中第 i 行的内容为输入矩阵的第 p_i 行. 假设重排列为一个函数 perm 即 $\boldsymbol{X}' = \mathrm{perm}(\boldsymbol{X})$, 已知梯度 $\frac{\partial \ell}{\partial \boldsymbol{X}'}$, 需要计算 $\frac{\partial \ell}{\partial \boldsymbol{X}}$. 对矩阵的行进行排列可以采用简单的循环实现, 例如plain_permutation_function中的实现方法.

```python
import numpy as np

def plain_permutation_function(X, p):
    # 初始化结果矩阵, 其中每一行对应一个样本
    permuted_X = np.zeros_like(X)
    for i in range(X.shape[0]):
        # 采用循环的方式对每一个样本进行重排列
        permuted_X[i] = X[p[i]]
    return permuted_X
```

请给出上述两种任务的向量化实现方案，并分析上述实现方法和向量化实现方法之间运行时间的差异。（提示：比如可以针对不同规模的矩阵大小来尝试分析主要操作的运行时间）

解：

1. 注意到

$$
\begin{aligned}
\|\boldsymbol{x}-\boldsymbol{y}\|_2 &= \sqrt{\sum_{i=1}^{d}(x_i-y_i)^2} \\
&= \sqrt{\sum_{i=1}^{d}x_i^2 + \sum_{i=1}^{d}y_i^2 - 2\sum_{i=1}^{d}x_iy_i} \\
&= \sqrt{\boldsymbol{x}^\top\boldsymbol{x} + \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{x}^\top\boldsymbol{y}}
\end{aligned}
$$

而输入矩阵 $\boldsymbol{X}$ 满足

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m)$$

因此对其做外积可以得到

$$\boldsymbol{M} = \boldsymbol{X}^\top \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \boldsymbol{x}_1 & \boldsymbol{x}_1^\top \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_1^\top \boldsymbol{x}_m \\ \boldsymbol{x}_2^\top \boldsymbol{x}_1 & \boldsymbol{x}_2^\top \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_2^\top \boldsymbol{x}_m \\ \vdots & \vdots & & \vdots \\ \boldsymbol{x}_m^\top \boldsymbol{x}_1 & \boldsymbol{x}_m^\top \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_m^\top \boldsymbol{x}_m \end{pmatrix}$$

因此通过矩阵 $\boldsymbol{M}$ 的元素可以很快表示出所求

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = \sqrt{\boldsymbol{M}_{ii} + \boldsymbol{M}_{jj} - 2\boldsymbol{M}_{ij}}$$

代码实现如下

```python
import numpy as np
from numpy import random as rd
import time


def plain_distance_function(X):
    # 直观的距离计算实现方法
    # 首先初始化一个空的距离矩阵D
    D = np.zeros((X.shape[0], X.shape[0]))
    # 循环遍历每一个样本对
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            # 计算样本i和样本j的距离
            D[i, j] = np.sqrt(np.sum((X[i] - X[j])**2))
    return D


def distance_function(X):
    D = np.zeros((X.shape[0], X.shape[0]))
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            tmp = X[i] - X[j]
            D[i, j] = np.sqrt(np.dot(tmp.T, tmp))
    return D


def distance_function_two(X):
    # ||x-y||_2 = sqrt(x^Tx + y^Ty - 2x^Ty)
    D = np.zeros((X.shape[0], X.shape[0]))
    XXT = np.dot(X, X.T)
    #                       [[x1Tx1, x1Tx2, ... x1Txm]
    #               XXT =    [x2Tx1, x2Tx2, ... x2Txm]
    #                        [...      ...        ... ]
    #                        [xmTx1, xmTx2, ... xmTxm]]
    for i in range(X.shape[0]):
        for j in range(X.shape[0]):
            D[i, j] = np.sqrt(XXT[i, i] + XXT[j, j] - 2 * XXT[i, j])
    return D
```

其中 distance_function 为仅仅将分量和改写为内积形式的优化，而 distance_function_two 为整体优化
在朴素的实现中，耗时最大的在循环体中对向量进行内积，采用的分量和的形式使得计算机无法对其进行优化（仅仅将分量和改写为内积形式，计算起来也会有优化）。而在新实现中，最耗时间的部分应为求解 $\boldsymbol{M}$ 的过程，这里可以利用计算机对矩阵计算的优化。其余部分（开根号）耗时相同。
观察到：当 times=1000，输入矩阵为 $\mathbb{R}^{30\times 20}$ 时，

$$
\begin{aligned}
T_{\text{plain}} &= 0.0005779334746999894 \\
T_{\text{new}} &= 0.0002907969126000125 \\
T_{\text{new\_two}} &= 0.00014920310349998545
\end{aligned}
$$

发现经过优化后恰好为各提升了 50% 的效率

2. 起初的想法为：用给出的 p 生成一个行变换矩阵 $\boldsymbol{K}$，满足

$$
\begin{cases}
k_{i,p[i]} &= 1 \\
k_{i,j} &= 0, j \neq p[i]
\end{cases}
$$

由于定理：

矩阵行变换等价于左乘行变换矩阵

因此将 $\boldsymbol{K}$ 左乘于原矩阵即可得到结果。
实际运行中发现：由于生成行变换矩阵仍然需要遍历所有的 p 中元素，而且由于最后还有一个耗时较大的矩阵相乘，因此最终测试出的运行时间反而略高于朴素实现。
后来改为采用 python 内置的矩阵切片方法，较为简易地实现了本题的目的。以下为本题涉及的所有代码

```python
import numpy as np
import pylab as p
from numpy import random as rd
import time

def plain_permutation_function(X, p):
    # 初始化结果矩阵，其中每一行对应一个样本
    permuted_X = np.zeros_like(X)
    for i in range(X.shape[0]):
        # 采用循环的方式对每一个样本进行重排列
        permuted_X[i] = X[p[i]]
    return permuted_X

def permutation_function(X, p):
```

```python
    trans = np.zeros_like(X)
    for i in range(X.shape[0]):
        trans[i][p[i]] = 1
    return np.dot(trans, X)


def permutation_function_two(X, p):
    return X[p, :]
```

其中

permutation_function : 为起初的想法，也即左乘行变换矩阵方法
permutation_function_two : 为采用 python 矩阵切片的方法

测试发现，当输入矩阵为 $\mathbb{R}^{20\times 20}$，测试运行一百万次时，得到的输出数据为

$$T_{\text{plain}} = 0.0011363053186999423$$
$$T_{\text{new}} = 0.0017210200120000082$$
$$T_{\text{new\_two}} = 0.0003125037785999666$$

容易发现，使用切片的方法效率显著提升（70%），而左乘行变换矩阵效果不明显（略差于原方法）。
并且当输入数据提升为 $\mathbb{R}^{30\times 30}$ 时，得到新输出时间为

$$T_{\text{plain}} = 0.0014645297344000028$$
$$T_{\text{new}} = 0.0032563880517999678$$
$$T_{\text{new\_two}} = 0.0003366874188000 1107$$

发现行变换矩阵方法效率显著降低，而切片方法发挥稳定。

三. (20 points) **支持向量机**

考虑标准的 SVM 优化问题如下 (即教材公式 (6.35)),

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi_i} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y_i\left(\boldsymbol{w}^\top \boldsymbol{x}_i + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, i \in [m]. \end{aligned} \tag{5}$$

注意到, 在 (2.1) 中, 对于正例和负例, 其在目标函数中分类错误的“惩罚”是相同的. 在实际场景中, 很多时候正例和负例错分的“惩罚”代

价是不同的（参考教材 2.3.4 节）. 比如考虑癌症诊断问题, 将一个确实患有癌症的人误分类为健康人, 以及将健康人误分类为患有癌症, 产生的错误影响以及代价不应该认为是等同的. 所以对负例分类错误的样本 (即 false positive) 施加 $k>0$ 倍于正例中被分错的样本的“惩罚”. 对于此类场景下

1. 请给出相应的 SVM 优化问题.
2. 请给出相应的对偶问题, 要求详细的推导步骤, 如 KKT 条件等.

解:

1. 由于 y_i 为取值在 $\{-1,1\}$ 的标记，我们可以构造

$$p=\frac{1-y_i}{2}\in\{0,1\}$$

当且仅当 y_i 为负例时取值为 1，因此所有真负例的损失之和为

$$Cp(\sum_{i=1}^{m}\xi_i)$$

而真正例的损失之和为

$$C(1-p)(\sum_{i=1}^{m}\xi_i)$$

对负例分类错误的惩罚加倍，则有新优化问题:

$$\begin{aligned}\min_{\boldsymbol{w},b,\xi_i}\quad & \frac{1}{2}\|\boldsymbol{w}\|^2+(Ckp+(1-p))\sum_{i=1}^{m}\xi_i\\ \text{s.t.}\quad & y_i\left(\boldsymbol{w}^\top\boldsymbol{x}_i+b\right)\geq 1-\xi_i\\ & \xi_i\geq 0, i\in[m]\\ & p=\frac{1-y_i}{2}.\end{aligned}$$

2. 加入 Lagrange 乘子 $\alpha_i\geq 0,\mu_i\geq 0$ 得到

$$\begin{aligned}L(\boldsymbol{w},b,\alpha,\xi,\mu)=&\frac{1}{2}\|\boldsymbol{w}\|^2+(Ckp+(1-p))\sum_{i=1}^{m}\xi_i\\ &+\sum_{i=1}^{m}\alpha_i(1-\xi_i-y_i(\boldsymbol{w}^\top\boldsymbol{x}_i+b))-\sum_{i=1}^{m}\mu_i\xi_i\end{aligned}$$

令 $L(\boldsymbol{w}, b, \alpha, \xi, \mu)$ 对 $\boldsymbol{w}, b, \xi_i$ 的偏导为 0 可得

$$\begin{aligned}
\boldsymbol{w} &= \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i \\
0 &= \alpha_i y_i \\
C &= \frac{1}{kp}(\alpha_i + \mu_i + p - 1) = \frac{2\alpha_i + 2\mu_i - y_i - 1}{k(1-y_i)}
\end{aligned}$$

将上式代回到 $L(\boldsymbol{w}, b, \alpha, \xi, \mu)$，则得到

$$\frac{1}{2}\|\boldsymbol{w}\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

$$(Ckp + (1-p))\sum_{i=1}^{m}\xi_i = \sum_{i=1}^{m}(\alpha_i + \mu_i)\xi_i$$

$$\sum_{i=1}^{m}\alpha_i(1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) = \sum_{i=1}^{m}\alpha_i(1-\xi_i) - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

因此有

$$\begin{aligned}
L(\boldsymbol{w}, b, \alpha, \xi, \mu) &= \frac{1}{2}\|\boldsymbol{w}\|^2 + (Ckp + (1-p))\sum_{i=1}^{m}\xi_i \\
&\quad + \sum_{i=1}^{m}\alpha_i(1 - \xi_i - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) - \sum_{i=1}^{m}\mu_i\xi_i \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{m}(\alpha_i + \mu_i)\xi_i + \sum_{i=1}^{m}\alpha_i(1-\xi_i) \\
&\quad - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_{i=1}^{m}\mu_i\xi_i \\
&= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j
\end{aligned}$$

因此得到相应的对偶问题为

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j \\
\text{s.t.} \quad & \alpha_i \geq 0, i \in [m]
\end{aligned}$$

KKT 条件为

$$\begin{cases}\alpha_i \geq 0, \mu_i \geq 0 \\ y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 + \xi_i \geq 0 \\ \alpha_i(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 + \xi_i) = 0 \\ \xi_i \geq 0, \mu_i \xi = 0\end{cases}$$

四. (20 points) **核函数**

教材 6.3 节介绍了 Mercer 定理, 说明对于一个二元函数 $k(\cdot,\cdot)$, 当且仅当对任意 m 和 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, 它对应的 Gram 矩阵 (核矩阵) 是半正定的时, 它是一个有效的核函数. 核矩阵 $\boldsymbol{K}$ 中的元素为 $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. 请根据 Mercer 定理证明以下核函数是有效的.

1. $\kappa_3 = a_1\kappa_1 + a_2\kappa_2$, 其中 $a_1, a_2 \geq 0$.
2. $f(\cdot)$ 是任意实值函数, 由 $\kappa_4(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x}) f(\boldsymbol{x}')$ 定义的 κ_4.
3. 由 $\kappa_5(\boldsymbol{x}, \boldsymbol{x}') = \kappa_1(\boldsymbol{x}, \boldsymbol{x}')\kappa_2(\boldsymbol{x}, \boldsymbol{x}')$ 定义的 κ_5.
4. 由 $\kappa_6(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})\kappa_1(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}')$ 定义的 κ_6

解:

1. 已知 κ_1, κ_2 为核函数，则为对称函数，则有

$$\begin{aligned}\kappa_3(x_i, x_j) &= a_1\kappa_1(x_i, x_j) + a_2\kappa_2(x_i, x_j) \\ &= a_1\kappa_1(x_j, x_i) + a_2\kappa_2(x_j, x_i) \\ &= \kappa_3(x_j, x_i)\end{aligned}$$

因此 κ_3 为对称函数

由 K_1, K_2 为半正定的，则对任意向量 $\boldsymbol{u} \in \mathbb{R}^m$，我们有

$$\boldsymbol{u}^\top K_1 \boldsymbol{u} \geq 0$$
$$\boldsymbol{u}^\top K_2 \boldsymbol{u} \geq 0$$

设 κ_3 的核矩阵为 K_3，则有

$$\begin{aligned}K_{3_{ij}} = \kappa_3(\boldsymbol{x}_i, \boldsymbol{x}_j) &= a_1\kappa_1(\boldsymbol{x}_i, \boldsymbol{x}_j) + a_2\kappa_2(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= a_1 K_{1_{ij}} + a_2 K_{2_{ij}}\end{aligned}$$

因此有

$$K_3 = a_1 K_1 + a_2 K_2$$

则对任意向量 $\boldsymbol{u} \in \mathbb{R}^m$，我们有

$$\begin{aligned}\boldsymbol{u}^\top K_3 \boldsymbol{u} &= \boldsymbol{u}^\top (a_1 K_1 + a_2 K_2)\boldsymbol{u} \\ &= a_1 \boldsymbol{u}^\top K_1 \boldsymbol{u} + a_2 \boldsymbol{u}^\top K_2 \boldsymbol{u} \\ &\geq 0\end{aligned}$$

因此 K_3 也为半正定的，由 Mercer 定理知 κ_3 是有效的

2. 由

$$\begin{aligned}\kappa_4(x_i, x_j) &= f(x_i)f(x_j) \\ &= f(x_j)f(x_i) \\ &= \kappa_4(x_j, x_i)\end{aligned}$$

知 κ_4 为对称函数，令其核矩阵为 K_4，则

$$K_4 = \begin{pmatrix} f(x_1)f(x_1) & f(x_1)f(x_2) & \cdots & f(x_1)f(x_m) \\ f(x_2)f(x_1) & f(x_2)f(x_2) & \cdots & f(x_2)f(x_m) \\ \vdots & \vdots & & \vdots \\ f(x_m)f(x_1) & f(x_m)f(x_2) & \cdots & f(x_m)f(x_m) \end{pmatrix}$$

将矩阵的第 2-m 行加到第一行上，得到

$$\begin{pmatrix} \sum_{i=1}^m f(x_i)f(x_1) & \sum_{i=1}^m f(x_i)f(x_2) & \cdots & \sum_{i=1}^m f(x_i)f(x_m) \\ f(x_2)f(x_1) & f(x_2)f(x_2) & \cdots & f(x_2)f(x_m) \\ \vdots & \vdots & & \vdots \\ f(x_m)f(x_1) & f(x_m)f(x_2) & \cdots & f(x_m)f(x_m) \end{pmatrix}$$

令 $k = \sum_{i=1}^m f(x_i), k_i = f(x_i)$，则原矩阵转化为有

$$\begin{pmatrix} kf(x_1) & kf(x_2) & \cdots & kf(x_m) \\ k_2 f(x_1) & k_2 f(x_2) & \cdots & k_2 f(x_m) \\ \vdots & \vdots & & \vdots \\ k_m f(x_1) & k_m f(x_2) & \cdots & k_m f(x_m) \end{pmatrix}$$

不难看出，当前矩阵的秩为 1，而初等变换不改变矩阵的秩，因此我们知道

$$rank(K_4) = 1$$

又 K_4 为对称矩阵，因此一定存在向量 $\boldsymbol{v} \in \mathbb{R}^m$ 满足

$$\boldsymbol{v}\boldsymbol{v}^\top = K_4$$

则对任意向量 $\boldsymbol{u} \in \mathbb{R}^m$，我们有

$$\boldsymbol{u}^\top K_4 \boldsymbol{u} = \boldsymbol{u}^\top \boldsymbol{v}\boldsymbol{v}^\top \boldsymbol{u} = (\boldsymbol{u}^\top \boldsymbol{v})(\boldsymbol{u}^\top \boldsymbol{v})^\top \geq 0$$

因此 K_4 为半正定的，由 Mercer 定理知 κ_4 是有效的

3. 由 κ_1, κ_2 是有效的核函数，则对任意向量 $\boldsymbol{u} \in \mathbb{R}^m$，我们有

$$\boldsymbol{u}^\top K_1 \boldsymbol{u} = \sum_{i=1}^{m}\sum_{j=1}^{m} \boldsymbol{u}_i \boldsymbol{u}_j K_{1_{ij}} \geq 0$$
$$\boldsymbol{u}^\top K_2 \boldsymbol{u} = \sum_{i=1}^{m}\sum_{j=1}^{m} \boldsymbol{u}_i \boldsymbol{u}_j K_{2_{ij}} \geq 0$$

并且由

$$\begin{aligned}\kappa_5(x_i, x_j) &= \kappa_1(x_i, x_j)\kappa_2(x_i, x_j) \\ &= \kappa_1(x_j, x_i)\kappa_2(x_j, x_i) \\ &= \kappa_5(x_j, x_i)\end{aligned}$$

知 κ_5 为对称函数，令其核矩阵为 K_5，则

$$K_5 = \begin{pmatrix} \kappa_1(x_1, x_1)\kappa_2(x_1, x_1) & \cdots & \kappa_1(x_1, x_m)\kappa_2(x_1, x_m) \\ \vdots & & \vdots \\ \kappa_1(x_m, x_1)\kappa_2(x_m, x_1) & \cdots & \kappa_1(x_m, x_m)\kappa_2(x_m, x_m) \end{pmatrix}$$

为 K_1, K_2 的 schur 乘积，由 schur 乘积定理知：当 K_1, K_2 半正定时，K_5 也是半正定的。因此由 Mercer 定理知 κ_5 为有效的核函数。

4. 由 κ_1 为有效的核函数，则有

$$\begin{aligned}\kappa_6(x_i, x_j) &= f(x_i)\kappa_1(x_i, x_j)f(x_j) \\ &= f(x_j)\kappa_1(x_i, x_j)f(x_i) \\ &= f(x_j)\kappa_1(x_j, x_i)f(x_i) \\ &= \kappa_6(x_j, x_i)\end{aligned}$$

由此知 κ_6 为对称函数，记其核矩阵为 K_6，令

$$F = \begin{pmatrix} f(x_1) & 0 & \cdots & 0 \\ 0 & f(x_2) & \cdots & 0 \\ \vdots & & \vdots & \\ 0 & 0 & \cdots & f(x_m) \end{pmatrix}$$

则容易得

$$K_6 = F^\top K_1 F$$

则对任意向量 $\boldsymbol{u} \in \mathbb{R}^m$，我们有

$$\begin{aligned} \boldsymbol{u}^\top K_6 \boldsymbol{u} &= \boldsymbol{u}^\top F^\top K_1 F \boldsymbol{u} \\ &= (F\boldsymbol{u})^\top K_1 (F\boldsymbol{u}) \\ &\geq 0 \end{aligned}$$

因此 K_6 为半正定的，由 Mercer 定理知 κ_6 是有效的

五. (20 points) **主成分分析**

$\boldsymbol{x} \in \mathbb{R}^d$ 是一个随机向量, 其均值和协方差分别是 $\boldsymbol{\mu} = \mathbb{E}(\boldsymbol{x}) \in \mathbb{R}^d$, $\boldsymbol{\Sigma} = \mathbb{E}(\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}})(\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}})^\top \in \mathbb{R}^{d\times d}$. 定义随机变量 $\{y_i = \boldsymbol{u}_i^\top \boldsymbol{x} + a_i \in \mathbb{R}, i = 1, \ldots, d' \leq d\}$ 为 $\boldsymbol{x}$ 的主成分, 其中 $\boldsymbol{u}_i \in \mathbb{R}^d$ 是单位向量 ($\boldsymbol{u}_i^\top \boldsymbol{u}_i = 1$), $a_i \in \mathbb{R}$, $\{y_i\}_{i=1}^{d'}$ 是互不相关的零均值随机变量, 它们的方差满足 $\mathrm{var}(y_1) \geq \mathrm{var}(y_2) \geq \cdots \geq \mathrm{var}(y_{d'})$. 假设 $\boldsymbol{\Sigma}$ 没有重复的特征值.

1. 请证明 $\{a_i = -\boldsymbol{u}_i^\top \boldsymbol{\mu}\}_{i=1}^{d'}$.
2. 请证明 $\boldsymbol{u}_1$ 是 $\boldsymbol{\Sigma}$ 最大的特征值对应的特征向量. (提示: 写出要最大化的目标函数, 写出约束条件, 使用拉格朗日乘子法)
3. 请证明 $\boldsymbol{u}_2^\top \boldsymbol{u}_1 = 0$，且 $\boldsymbol{u}_2$ 是 $\boldsymbol{\Sigma}$ 第二大特征值对应的特征向量. (提示: 由 $\{y_i\}_{i=1}^{d}$ 是互不相关的零均值随机变量可推出 $\boldsymbol{u}_2^\top \boldsymbol{u}_1 = 0$, 可作为第二小问的约束条件之一)
4. 通过 PCA 进行降维, 得到的随机变量满足 $\mathrm{var}(y_1) \geq \mathrm{var}(y_2) \geq \cdots \geq \mathrm{var}(y_d)$, 也就是降维后的数据在不同维度上有不同的方差, 从而导致不同维度的数值范围差异很大, 如果想要降维后的样本在不同维度具有大致相同的数值范围, 应该怎么做?

解:

1. 由
$$y_i = \boldsymbol{u}_i^\top \boldsymbol{x} + a_i$$
且
$$\mathbb{E}(\boldsymbol{x}) = \boldsymbol{\mu}$$
$$\mathbb{E}(y_i) = 0$$
知：对一式求均值有
$$\begin{aligned} \mathbb{E}(y_i) &= \mathbb{E}(\boldsymbol{u}_i^\top \boldsymbol{x} + a_i) \\ \iff 0 &= \boldsymbol{u}^\top \boldsymbol{\mu} + a_i \\ \iff a_i &= -\boldsymbol{u}^\top \boldsymbol{\mu} \end{aligned}$$
因此有
$$\{a_i = -\boldsymbol{u}_i^\top \boldsymbol{\mu}\}_{i=1}^{d'}$$

2. 由第 1 小问知
$$y_i = \boldsymbol{u}_i^\top \boldsymbol{x} + a_i = \boldsymbol{u}_i^\top (\boldsymbol{x} - \boldsymbol{\mu})$$
由 y_i 为零均值随机变量可得：
$$\begin{aligned} \operatorname{var}(y_i) &= E(y_i^2) - E(y_i)^2 = E(y_i y_i^\top) \\ &= E(\boldsymbol{u}_i^\top (\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{u}_i) \\ &= \boldsymbol{u}_i^\top \boldsymbol{\Sigma} \boldsymbol{u}_i \end{aligned}$$
可写出优化问题
$$\begin{aligned} \arg\max_i \quad & \boldsymbol{u}_i^\top \boldsymbol{\Sigma} \boldsymbol{u}_i \\ \text{s.t.} \quad & \boldsymbol{u}_i^\top \boldsymbol{u}_i = 1 \end{aligned}$$
引入 Lagrange 乘子可得
$$L(\boldsymbol{u}_i, \lambda) = \boldsymbol{u}_i^\top \boldsymbol{\Sigma} \boldsymbol{u}_i + \lambda(1 - \boldsymbol{u}_i^\top \boldsymbol{u}_i)$$
分别令该函数对两自变量求偏导为 0 可以得到
$$\frac{\partial L(\boldsymbol{u}_i, \lambda)}{\partial \boldsymbol{u}_i} = 2\boldsymbol{\Sigma} \boldsymbol{u}_i - 2\lambda \boldsymbol{u}_i = 0$$
$$\frac{\partial L(\boldsymbol{u}_i, \lambda)}{\partial \lambda} = \boldsymbol{u}_i^\top \boldsymbol{u}_i = 1$$

因此 λ 为 $\mathbf{\Sigma}$ 的特征值，并且我们得到

$$\lambda = \boldsymbol{u}_i^\top \mathbf{\Sigma} \boldsymbol{u}_i = \mathrm{var}(y_i)$$

由 $\mathrm{var}(y_1)$ 是最大的知其为 $\mathbf{\Sigma}$ 最大的特征值，因此 $\boldsymbol{u}_1$ 为 $\mathbf{\Sigma}$ 最大的特征值对应的特征向量。

3. 由 y_i 为互不相关的随机变量，以及 $\boldsymbol{u}_i$ 为单位向量可知，若 $\boldsymbol{u}_2^\top \boldsymbol{u}_1 \neq 0$, 则 $\boldsymbol{u}_2 = \boldsymbol{u}_1$, 则有 $y_1 = y_2$，这与条件 y_i 为互不相关的随机变量矛盾，因此有

$$\boldsymbol{u}_2^\top \boldsymbol{u}_1 = 0$$

由第 2 小问的计算可知，$\mathrm{var}(y_i)$ 的大小关系即为 $\mathbf{\Sigma}$ 的最大的 d' 个特征值的大小关系，且其特征向量为 $\boldsymbol{u}_i$

4. 可以通过对降维后的所有数值套用同一个 Sigmoid 函数变换到 [0,1] 这段区间中，以获得大致相同的数据范围。