北京大学信息学院考试试卷

考试科目：Python 与数据科学导论 **姓名：**_____ **学号：**_____.
考试时间：__202*__年__6__月____日　　**任课教师：**胡俊峰

题号	一	二	三	四	五	六	七	八	**总分**
分数									
阅卷人									

北京大学考场纪律

1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。

2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机、或有存储、编程、查询功能的电子用品等）不得带入座位，已经带入考场的必须放在监考人员指定的位置。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束时间到，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。

4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准交头接耳，不准偷看、夹带、抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有违纪作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学术规范条例》及相关规定严肃处理。

5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。

学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声

以下为试题和答题纸，共　　页。

一、 **多重选择题：（每小题 2 分，共 16 分）**
从候选答案中选择 1 到多个正确答案，每个选项占 0.5 分

1) 考虑 numpy 的 multiply 操作，对 x 和 y 两个 array 来说，以下哪些格式可以正常计算不会报错？
A. x: (1,1) y: (2,30)
B. x: (1,30) y: (5,30)
C. x: (1,5) y. (1,30)
D. x: (20,5) y: (5,20)
答案：（ ABC AB ）

2) 以下哪种训练技巧会使得模型参数变稀疏？
A. L2 正则化 B. L1 正则化 C. Dropout D. teacher forcing
答案：（ C B ）

4、常见的将非平稳序列转换成平稳序列的方法有：

A) 对原序列进行差分操作

B) 对原序列进行平方开放操作

C) 对原序列进行取对数运算

D) 对原序列将原序列进行滑动窗口平均操作

答案：（ ACD AD ）

6、仿关于离散余弦变换（DCT）和图像压缩，下列说法正确的是
A. DCT 能够将时序信号投影到频域特征空间
B. 图像处理中常用二维的 DCT，等价于在一维 DCT 的基础上再做一次 DCT
C. JPEG 是一种有损图像压缩算法，量化是造成损失的最主要原因
D. 在由量化步长构成的矩阵中，左上角对应高频分量，右下角对应低频分量，因为人眼对于低频分量更敏感，所以矩阵左上角的值普遍小于右下角
答案：（ AC ）

7、下关于协程(routine)，下面哪些说法是正确的？

A. 全局解释锁(GIL)保证任何时刻只有一个协程执行，因此在多协程之间不需要加锁。

B. 协程之间不是并发的关系

C. 每个协程有单独的 python 解释器实例执行指令

D. 协程常用于 I/O 通讯，资源管理与操作响应等

答案：（ BCD BD ）

8、关于 Kmeans 聚类算法，下列说法正确的是：

A. 可以看作是一种特殊的矩阵分解问题

B. 该算法属于监督学习

C. 最终的聚类结果与初始聚类中心的选择无关

D. 可以使用不同的距离函数和核函数

答案：（ D AD ）

二、阅读程序并给出运行结果（共 30 分）

```
import copy
ls = [1, 2, [3, 4]]
c = copy.copy(ls)
ls[-1].append(5)
ls.append(6)
print(ls)
print(c)
```

请写出上面程序的运行结果:

```
[1, 2, [3, 4, 5], 6]
[1, 2, [3, 4, 5]]
```

5、

```
def func_a(func_a_arg_a, func, **kwargs):
    print(func_a_arg_a)
    func(**kwargs)

def func_b(arg_a):
    print(arg_a)

def func_c():
    print('Hello World')

if __name__ == '__main__':
```

```
    func_a(func_a_arg_a='temp', arg_a='Hello Python', func=func_b)
    func_a(func_a_arg_a='temp', func=func_c)
```
请写出上面程序的运行结果：
temp
Hello Python
temp
Hello World

6、
```
def fun(items):
    se = set()
    for it in items:
        if it not in se:
            yield it
            se.add(it)
a = [1, 5, 2, 1, 9, 1, 5, 10]
print(list(fun(a)))
```
请写出上面程序的运行结果：
[1,5,2,9,10]

7、
```
mat_1 = [['a','b','c'],['d','e','f']]
mat_2 = ['1','2','3']
result = ','.join([i+j
   for vec in mat_1
   for i,j in zip(vec, mat_2)])
print(result)
```
请写出上面程序的运行结果：
a1

b2
c3
d1
e2
f3

8、

```
def call_Fun_counter(func):
    def wrapper(*args, **kwargs):
        wrapper.calls += 1
        return func(*args, **kwargs)
    wrapper.calls = 0
    return wrapper

@call_Fun_counter
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

print(fib(4))
print(fib.calls)
```

请写出上面程序的运行结果：

3

9

9、import numpy as np
a = np.array([[1,2,3],[2,3,4]])
b = a
b += 1
print(a)
a = a.T + b[0,1:]
print(a)
请写出上面程序的运行结果：
[[2 3 4]
 [3 4 5]]
[[5,7],
[6,8],
[7,9]]

10、

```
def main():
    try:
        func()
        print("function ends")
    except ZeroDivisionError:
        print('Divided By Zero! ')
    except:
        print('Its an Exception!')

def func():
    print(1/0)

main()
```

请写出上面程序的运行结果：
~~ZeroDivisionError: 'Divided By Zero!~~
Divided By Zero!

三、Python 代码填空（共 16 分）

用代码进行代码填空

(如果一行能完成，尽量写一行，多于一行代码视情况可能会扣 0.5-1 分)

1、请用列表表达式生成 50 以内 4 的倍数（2 分）

_____[I for I in range(50) if I % 4 == 0]

2、姓名和年龄的 list，请实现按照年龄排序，年龄相同再按姓名排序 （2 分）

```
lst = [["zs", 19], ["ll", 54],["wa",23], ["df", 23],["xf",23]]

lst2 = list(sorted(sorted(lst, lambda x:x[0]), lambda x:x[1]))
```

3、标准化是特征处理的常规方法，对于给定的矩阵 X，请按列对其进行标准化（x-mean/std）（2 分）

```
X = np.random.random((100, 30))

X_mean = _____np.mean(X,axis=0)_____(1 分)

X_std =  _____np.std(X,axis=0)_____(1 分)
```

4. 垃圾邮件中可能包含恶意的电子邮箱地址，请写出判断一个字符串是否为合法电子邮箱地址的正则表达式。为简化问题，假设电子邮箱必须有且仅有一个@，包含若干大小写字母、数字、短横线-和英文句点.，并且两个英文句点不能相邻。（2 分）

——[a-zA-Z0-9.-]————————————————

5. 考虑 CNN 的卷积操作，输入为 长 L*宽 L*通道数 c，卷积核大小为 长 k*宽 k，共有 m 个卷积核，填充为 p 步长 s，问输出层的尺寸（size）和通道数分别为？（2 分）

———(l+2p-k)/s——， ————m/c————

四、深度学习部分（10 分）

下面是一段使用 NumPy 搭建神经网络的代码，损失函数为交叉熵：

```
import numpy as np
def sigmoid(x):
    return    1/(1+np.exp(-x))

def forward(W_1, W_2, X, Y):
    z_2 = np.dot(X, W_1)
    a_2 = sigmoid(z_2)
    z_3 = np.dot(a_2,W_2)
    y_pred = sigmoid(z_3)

    J_z_3_grad =   sigmoid(y_pred)*(1- sigmoid(y_pred))
    J_W_2_grad = a_2.T @ J_z_3_grad
    J_a_2_grad = J_z_3_grad @ W_2.T
    a_2_z_2_grad =
    J_z_2_grad =
    J_W_1_grad =
        return y_pred, (J_W_1_grad, J_W_2_grad)
```

（1）（6 分）代码填空

（2）（2 分）在 MNIST 数据集的一个较小子集上使用该神经网络进行训练，发现产生了过拟合现象，写出两种合理的解决方式

（3）（2 分）训练一个二分类任务时，如果训练数据类别不平衡（正例较多，负例较少），写出两种合理的提高分类准确率的方法

答案：

```
import numpy as np


def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def forward(W_1, W_2, X, Y):
    z_2 = np.dot(X, W_1)
    a_2 = sigmoid(z_2)

    z_3 = np.dot(a_2, W_2)
    y_pred = sigmoid(z_3)

    J_z_3_grad = y_pred - Y
    J_W_2_grad = a_2.T @ J_z_3_grad
    J_a_2_grad = J_z_3_grad @ W_2.T
    a_2_z_2_grad = a_2 * (1 - a_2)
    J_z_2_grad = J_a_2_grad * a_2_z_2_grad
    J_W_1_grad = X.T @ J_z_2_grad

    return y_pred, (J_W_1_grad, J_W_2_grad)
```

五、简答题（共 12 分）：

2、简述协程概念以及 python 中有哪两种实现协程的机制？（4 分）

在 Python 中，`yield` 关键字在生成器函数中的使用与协程有关联。尽管 Python 的 `yield` 本身最初设计是用于创建生成器（generator），但它也被广泛用于实现协程。

协程是一种能够暂停和恢复执行的函数或子程序，与传统的函数调用不同，协程可以在执行过程中暂停，并将控制权交还给调用者，然后再次从暂停的地方继续执行。这种交替执行的方式使得协程可以在一个线程内实现并发和异步操作。

在 Python 中，通过生成器函数中的 `yield` 关键字，我们可以创建协程。`yield` 的作用是产生一个值并暂停函数的执行，同时允许外部代码向协程发送值。当协程被恢复执行时，它会从上一次暂停的地方继续执行，且可以接收外部发送的值。这种交互式的执行方式使得协程能够实现复杂的控制流和协作式的多任务处理。

Python 3.5 引入的 `async` 和 `await` 关键字进一步扩展了协程的概念，并提供了更方便的语法来定义和使用协程。`async` 用于定义一个协程函数，而 `await` 用于挂起协程并等待其结果。

总结来说，Python 的 `yield` 关键字在生成器函数中的使用为实现协程提供了基础，通过使用 `yield`，我们可以创建可以暂停和恢复的函数，从而实现协程的特性。随着 Python 3.5 的引入，协程的概念进一步扩展，提供了更强大的异步编程能力。

3、用 LSTM+attention 机制实现的序列生成模型（seq2seq），在实际使用中在输出序列中会容易生成一些重复的单词。请简要分析这种现象的原因（2 分），给出你认为合理的解决问题方案（2 分）

生成重复单词的现象在使用 LSTM+attention 机制实现的序列生成模型中是常见的。这种现象通常是由于模型的局限性和训练数据的限制引起的。

原因分析：
1. 缺乏全局信息：LSTM+attention 模型在生成每个单词时，通常只关注输入序列的局部信息和当前生成的上下文，而缺乏对全局语义的把握。这可能导致模型重复生成之前已经生成过的单词。

2. 数据偏差：如果训练数据中存在偏差或重复的样本，模型可能会学习到这种模式并重复生成相似的单词。

解决问题的方案：
1. Beam Search：在生成阶段使用 Beam Search 算法，而不是仅依赖于单个生成的结果。Beam Search 会保留多个备选的生成序列，并根据预定义的评分函数选择最优的生成序列。通过扩展搜索空间，可以减少重复生成的可能性。

2. 标记限制：引入一种机制来标记已经生成过的单词，以防止模型重复生成相同的单词。可以使用一个词汇表来记录已生成的单词，并在生成新单词时进行检查。

3. 多样性增强：通过引入随机性或噪声，以及在训练过程中采用不同的数据增强技术，可以增加模型生成序列的多样性。例如，使用 dropout 技术，在训练过程中随机丢弃部分神经元，增加模型的随机性。

4. 更多训练数据：通过增加训练数据量，包括更多的不同样本和语义上多样化的数据，可以提供更丰富的信息和更全面的语义表示，从而减少生成重复单词的可能性。

这些解决方案可以结合使用，根据具体情况进行调整和优化。重点是在生成阶段引入更多的控制和多样性，同时确保生成结果的语义准确性和流畅性。

函数实现：（共 16 分）

1、下面给出了一个二叉树的类型定义

```
class BinaryTree(object):
    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None

    def insertLeft(self,newNode):
        if self.leftChild == None:
            self.leftChild = BinaryTree(newNode)
        else:
            t = BinaryTree(newNode)
            t.leftChild = self.leftChild
            self.leftChild = t

    def insertRight(self,newNode):
        if self.rightChild == None:
            self.rightChild = BinaryTree(newNode)
        else:
            t = BinaryTree(newNode)
            t.rightChild = self.rightChild
            self.rightChild = t

    def getRightChild(self):
        return self.rightChild

    def getLeftChild(self):
        return self.leftChild
    def setRootVal(self,obj):
        self.key = obj

    def getRootVal(self):
  return self.key
```

要求：

1）写出语句序列生成一个该类型的实例 r，包含 3 个结点，根节点内容为字符串“+“，左子树节点内容为字符串“15”，右子树内容为字符串“10”（2 分）
语句序列：

2）为这个 BinaryTree 类添加一个成员函数 countLeaf 方法，实现对实例中节点数的计数，并返回计数值。比如上面那个树的实例，调用该方法返回值为 3（2 分）
语句序列（包含函数定义和添加成员函数到类中的语句）：

2、下面是一个可以正常执行的代码环境的部分代码，要求：
1）在空白处补充 numpy 代码，实现用卷积核进行图像边缘提取的操作（8 分）
2）给出代码中两条 print 语句的输出结果（2 分）

```
import numpy as np
lena = cv2.imread('pic/lena.jpg',cv2.IMREAD_GRAYSCALE)
print(lena.shape)
```

```
(512, 512)
```

```
def convolution(image , kernel):
    res = np.zeros(image.shape,np.uint8)
    h , w = kernel.shape
    image = np.pad(image, ((h//2,h//2), (w//2,w//2)), 'median')
    h_image , w_image = image.shape
#补充代码用kernal对输入图片进行步长为1的卷积,实现图像增强
    print(res.shape)
    print(image.shape)
    return res
kernel = np.array([
        [-1, -1, -1],
        [-1, 8, -1],
        [-1, -1, -1]
    ])
res = convolution(lena,kernel)
```

在这里给出上面代码中两条 print 语句的输出结果：

3.在机器翻译任务使用的基于 LSTM 的 seq2seq 递归网络模型中，经常会使用 attention 机制进行结果优化。同时在结果生成中会采用 beam search 算法。
请问 beam search 算法的简单流程：

Beam Search 是一种用于在序列生成任务中进行搜索和解码的算法。它通常应用于机器翻译、语言生成和序列到序列模型等任务中。

在序列生成任务中，例如机器翻译，模型需要根据输入序列生成输出序列，而 Beam Search 可以帮助选择最优的输出序列。

Beam Search 算法的基本思想是在生成过程中维护一个由多个备选序列组成的集合，这些序列称为 Beam。在每个时间步，根据模型生成的概率分布，从当前 Beam 中的每个序列扩展出多个备选序列。然后根据预定义的评分函数，从这些备选序列中选择前 k 个分数最高的序列作为新的 Beam。

这个过程会不断进行，直到达到生成序列的最大长度或满足特定停止条件。最终，Beam 中的序列中的最高分序列即为最终的输出序列。

Beam Search 的优点是能够保留多个备选序列，从而扩展搜索空间，减少因局部最优解而错过全局最优解的可能性。它能够在保证一定解码速度的情况下，提供相对较好的解码结果。

需要注意的是，Beam Search 也有一些限制。例如，由于它的局部性质，它可能会受到搜索空间大小 k 的限制，无法保证找到全局最优解。此外，Beam Search 在搜索过程中可能会导致重复的短语或单词出现在生成序列中。

为了解决 Beam Search 中的一些问题，还可以结合其他技术，如长度惩罚、重复惩罚和多样性增强等，以提高生成结果的质量和多样性。