目录

第四十二篇：10 月 11 日-叮咚买菜-机器学习工程师面试题 -- 4 道

42.1 解释一下什么是 boosting 模型，以及 boosting 模型有哪几种

1、常见的 boosting 模型有 adaboost, gbdt, xgboost

2、boosting 算法是由多个树模型通过叠加的方式来学习特征和标签之间的规律

3、虽然有很多的 boosting 算法，其核心思想是使用新的模型去拟合之前所有叠加起来的模型所没有捕捉到的规律，从而一点点的学习特征和标签的规律

42.2 adaboost 分类模型和 boosting 分类模型的异同是什么

相同点：

1、两者都是 boosting 模型

2、两者都是使用新的模型去学习之前所有模型所没捕捉到的规律

不同点：

1、两者的损失函数不一样，adaboost 分类用的损失函数指数损失函数，而 gbdt 分类用的损失函数是 logloss 并且以 logodds 做的计算

2、两个的基模型不一样，adaboost 分类的基是分类树模型，gbdt 分类的基是回归树模型。

3、两个模型中，新模型学习之前模型所没有捕捉到的规律的方式不一样，adaboost 中，之前的模型通过加大预测错的样本的权重，从而提醒下一个模型要认真学习预测错的样本(之前所没有捕捉到的规律)。而 GBDT 中，直接通过 logodds 的残差值来训练新的模型

42.3 gbdt 如何做回归任务

1、基模型为回归树模型

2、损失函数为 MSE

3、使用新的模型去拟合之前所有的回归树叠加起来下遗留下来的残差，从而学习特征和标签之间的规律

42.4 gbdt 和 xgboost 的区别是什么?

1、xgboost 可以看成 gbdt 的升级版

2、xgboost 其实就是在 gbdt 的损失函数上加了约束项，从而控制整体模型的复杂度。

3、约束项为新的树模型的叶节点个数相关，和叶节点输出值相关。

4、xgboost 包算法不仅仅可以做分类，回归，而且可以做排序

5、xgboost 包可以部署分布式

第四十三篇：10 月 11 日-货拉拉-NLP 工程师面试题 -- 6 道

43.1 词向量平均法做分类的优劣势是什么

优势

1、词向量平均的方法做分类模型，主要的优势是模型简单

2、有参数模型，无参数模型都可以尝试使用，模型选择大

3、模型速度极快，训练的参数量少

4、在语句少的场景下，效果好

劣势

1、在语句长的长的场景下，效果会变的很差

2、语句长，分出的词多，词越多，信息量越杂，简单的做平均的话，重要的词的信息会在平均的过程中极大的被消弱，从而分类效果差

43.2 词向量的基础上如何做优化

1、映入一个新的向量，做 attention，此向量专门对重要的，和标签相关的词敏感。从而通过加权平均的方式，得到的句向量只包含重要词的信息，忽略不重要的词的信息，从而加强模型的效果。

2、使用 self-attention, 尝试对语句里词的分布做重新的调整，提高模型的学习能力

3、使用 Transformer encoder 或者 bert 来做学习

43.3 Bert 模型和 Transformer 模型之间的关系

1、Transformer 模型有 encoder 和 decoder

2、Bert 其实就是 Transformer 的 encoder 的部分

3、Transformer 只是一个空模型，里面的参数都是随机的，需要在下游任务上做有监督的训练，由于参数量大，直接使用 Transformer 做训练，模型难收敛，并且速度慢

4、Bert 其实是预训练好的 Transformer 的 encoder 部分，也就是已经在海量的数据集上做了 Transformer 的参数的训练了，其参数可以保存下来，直接拿来在下游任务上使用，做调优。

43.4 Bert 模型中有哪些预训练的任务

1、首先是 MLM 任务，masked language model。随机的 mask 掉一些词，从而基于上下文，通过 attention 的方法来训练，来预测 mask 的词。

2、NSP 任务，Next Sentence Prediction 任务。通过 CLS 的来进行二分类，查看当前两个句子是否是上下文。词任务的有效性在各大论文中也是有争议的，有的说这个任务有用，有的说这个任务没用。

43.5 Bert 模型的做句向量的缺陷

1、直接使用 Bert 做句向量的输出，会发现所有的句向量两两相似度都很高。

2、因为对于句子来说，大多数的句子都是使用常见的词组成的。

3、Bert 的词向量空间是非凸的，大量的常见的词向量都是在 0 点附近，从而计算出的句子向量，都很相似。

43.6 如何解决 Bert 句向量的缺陷

1、使用双卡的形式，将两个句子传入两个参数共享的 Bert 模型，将两个句向量做拼接，进行有监督的学习，从而调整 Bert 参数。此方法叫 sentencebert。

2、使用无监督或者有监督的对比学习，将同一个句子传入相同的 bert(dropout = 0.3)得到标签为正例的一个句子对。通过这种方式来做 Bert 的微调整，得到 SimCSE 模型。

第四十四篇：2021 年 9 月底--百度 NLP 岗位面试题 2 道

44.1 先序遍历（要求递归和迭代两种方式）

方法一：递归

树本身就有递归的特性，因此递归方法最简单，这里直接放上代码，需要说明的是，中序遍历，前序遍历和后序遍历可采用相同的代码模板完成实现。

```
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if not root:
            return []
        return [root.val] + self.preorderTraversal(root.left) + self.preorderTraversa
l(root.right)
```

时间复杂度：O(n)，n 为树的节点个数

空间复杂度：O(h)，h 为树的高度

方法二：迭代

代码如下：

```
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        if not root:
            return []
        stack = []
        res = []
        cur = root
        while stack or cur:
            while cur:
                stack.append(cur)
                res.append(cur.val)
                cur = cur.left
            cur = stack.pop()
            cur = cur.right
        return res
```

时间复杂度：O(n)，n 为树的节点个数

空间复杂度：O(h)，h 为树的高度

44.2 旋转数组寻找 k

思路一：暴力解法

直接遍历整个数组，找到目标值 target

代码如下：

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        for i,num in enumerate(nums):
            if num == target:
                return i
        return -1
```

时间复杂度：O(n)

空间复杂度：O(1)

思路二：二分查找

先要设置整个数组的左右两端端点：left = 0，right = len(nums) - 1

1、若 target == nums[mid]，直接返回

2、若 nums[left] <= nums[mid]，说明左侧区间 [left,mid]「连续递增」。此时：

若 nums[left] <= target <= nums[mid]，说明 target 位于左侧。令 right = mid-1，在左侧区间查找

否则，令 left = mid+1，在右侧区间查找

3、否则，说明右侧区间 [mid,right]「连续递增」。

此时：

若 nums[mid] <= target <= nums[right]，说明 target 位于右侧区间。令 left = mid+1，在右侧区间查找

否则，令 right = mid-1，在左侧区间查找

代码如下：

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) -1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target:
                return mid
            elif nums[left] <= nums[mid]:
```

```
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

时间复杂度：O(logn)

空间复杂度：O(1)

第四十五篇 2021 年 9 月底--字节跳动 NLP 岗位（抖音）面试题 -- 8 道

45.1 Bert 模型中，根号 dk 的作用

QK 进行点击之后，值之间的方差会较大，也就是大小差距会较大；如果直接通过 Softmax 操作，会导致大的更大，小的更小；进行缩放，会使参数更平滑，训练效果更好。

45.2 Bert 模型中多头的作用

多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷积核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。

45.3 BPE 的了解

BPE 与 Wordpiece 都是首先初始化一个小词表，再根据一定准则将不同的子词合并。词表由小变大

BPE 与 Wordpiece 的最大区别在于，如何选择两个子词进行合并：BPE 选择频数最高的相邻子词合并，而 WordPiece 选择能够提升语言模型概率最大的相邻子词加入词表。

45.4 mask 策略和改进

从 bert 最开始的 mask token 到后面 ernie 的 mask entity 以及还有 mask n-gram，动态 mask 等等。

45.5 Bert 模型中激活函数 GELU

GELU 函数是 RELU 的变种，形式如下：

$$GELU(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}[1 + \text{erf}(x/\sqrt{2})]$$

45.6 部分激活函数的公式及求导

常见的激活函数有：Sigmoid、Tanh、ReLU

Sigmoid 函数：

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

求导：

$$\begin{aligned}\sigma'(x) &= \left(\frac{1}{1+e^{-x}}\right)' \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\ &= \sigma(x)(1-\sigma(x))\end{aligned}$$

特点：

它能够把输入的连续实值变换为 0 和 1 之间的输出，特别的，如果是非常大的负数，那么输出就是 0；如果是非常大的正数，输出就是 1。

缺点：

(1) 缺点 1：在深度神经网络中梯度反向传递时导致梯度消失，其中梯度爆炸发生的概率非常小，而梯度消失发生的概率比较大。

(2) 缺点 2：Sigmoid 的 output 不是 0 均值（即 zero-centered）。

(3) 缺点 3：其解析式中含有幂运算，计算机求解时相对来讲比较耗时。对于规模比较大的深度网络，这会较大地增加训练时间。

Tanh 函数：

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

求导：

$$tanh(x)' = \frac{(e^x+e^{-x})^2 - (e^x-e^{-x})^2}{(e^x+e^{-x})^2} = 1-(tanh(x))^2$$

特点：它解决了 Sigmoid 函数的不是 zero-centered 输出问题，收敛速度比 sigmoid 要快，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。

ReLU 函数：

$$ReLU(x) = \begin{cases} 0, & x \leqslant 0 \\ x, & x > 0 \end{cases}$$

求导：

$$ReLU(x)' = \begin{cases} 0, & x \leqslant 0 \\ 1, & x > 0 \end{cases}$$

特点：

(1) ReLu 函数是利用阈值来进行因变量的输出，因此其计算复杂度会比剩下两个函数低（后两个函数都是进行指数运算）

(2) ReLu 函数的非饱和性可以有效地解决梯度消失的问题，提供相对宽的激活边界。

(3) ReLU 的单侧抑制提供了网络的稀疏表达能力。

45.7 最短矩阵路径和

该题为 leetcode 第 64 题

思路：动态规划

由于路径的方向只能是向下或向右，因此网格的第一行的每个元素只能从左上角元素开始向右移动到达，网格的第一列的每个元素只能从左上角元素开始向下移动到达，此时的路径是唯一的，因此每个元素对应的最小路径和即为对应的路径上的数字总和。

对于不在第一行和第一列的元素，可以从其上方相邻元素向下移动一步到达，或者从其左方相邻元素向右移动一步到达，元素对应的最小路径和等于其上方相邻元素与其左方相邻元素两者对应的最小路径和中的最小值加上当前元素的值。由于每个元素对应的最小路径和与其相邻元素对应的最小路径和有关，因此可以使用动态规划求解。

创建二维数组 dp，与原始网格的大小相同，dp[i][j] 表示从左上角出发到 (i,j)位置的最小路径和。显然，dp[0][0]=grid[0][0]。对于 dp 中的其余元素，通过以下状态转移方程计算元素值。

- 当 $i > 0$ 且 $j = 0$ 时，$dp[i][0] = dp[i-1][0] + grid[i][0]$。
- 当 $i = 0$ 且 $j > 0$ 时，$dp[0][j] = dp[0][j-1] + grid[0][j]$。
- 当 $i > 0$ 且 $j > 0$ 时，$dp[i][j] = \min(dp[i-1][j], dp[i][j-1]) + grid[i][j]$。

代码如下：

```
1.  class Solution:
2.      def minPathSum(self, grid: List[List[int]]) -> int:
3.          m = len(grid)
4.          n = len(grid[0])
5.          dp = [[0] * n for _ in range(m)]
6.          dp[0][0] = grid[0][0]
7.          for i in range(1, m):
8.              dp[i][0] = dp[i-1][0] + grid[i][0]
9.          for j in range(1, n):
10.             dp[0][j] = dp[0][j-1] + grid[0][j]
```

```
        for i in range(1, m):
            for j in range(1,n):
                dp[i][j] = min(dp[i-1][j],dp[i][j-1]) + grid[i][j]
        return dp[-1][-1]
```

时间复杂度：O(mn)

空间复杂度：O(mn)

45.8 有序的全排列

思路：回溯算法

使用回溯算法框架即可，先考虑一位的情况，然后把递归地考虑去掉这一位数后的数组的全排列。

代码如下：

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        result = []
        def backtrack(nums,tmp):
            if not nums:
                result.append(tmp)
                return
            for i in range(len(nums)):
                backtrack(nums[:i] + nums[i+1:],tmp + [nums[i]])
        backtrack(nums,[])
        return result
```

时间复杂度：O(n×n!)

空间复杂度：O(n)

第四十六篇：2021 年 9 月中旬--百度 NLP 岗位 面试题 -- 10 道

46.1 神经网络有哪些初始化的方法、为什么要初始化

不初始化可能会减慢收敛速度，影响收敛效果。

如果使用的是预训练模型，是不需要自己进行初始化的，只有没有预训练模型的时候需要初始化。

常用的权重初始化算法是 「kaiming_normal」 或者 「xavier_normal」。

以下n_in为网络的输入大小,n_out为网络的输出大小，n为n_in或$(n_in + n_out) * 0.5$

- uniform均匀分布初始化：

$$w = np.random.uniform(low = -scale, high = scale, size = [n\_in, n\_out])$$

- Xavier初始法，适用于普通激活函数(tanh，sigmoid)：

$$scale = np.sqrt(3/n)$$

- He初始化，适用于ReLU：

$$scale = np.sqrt(6/n)$$

- normal高斯分布初始化， 其中stdev为高斯分布的标准差，均值设为0：

$$w = np.random.randn(n\_in, n\_out) * stdev$$

- Xavier初始法，适用于普通激活函数 (tanh,sigmoid)：

$$stdev = np.sqrt(n)$$

- He初始化，适用于ReLU：

$$stdev = np.sqrt(2/n)$$

- svd初始化：对RNN有比较好的效果。参考论文：https://arxiv.org/abs/1312.6120[8]

46.2 python 迭代器和生成器

可迭代对象（Iterable Object），简单的来理解就是可以使用 for 来循环遍历的对象。比如常见的 list、set 和 dict。

可迭代对象具有__iter__ 方法，用于返回一个迭代器，或者定义了 getitem 方法，可以按 index 索引的对象（并且能够在没有值时抛出一个 IndexError 异常），因此，可迭代对象就是能够通过它得到一个迭代器的对象。所以，可迭代对象都可以通过调用内建的 iter() 方法返回一个迭代器。

生成器其实是一种特殊的迭代器，不过这种迭代器更加优雅。它不需要再像上面的类一样写__iter__()和__next__()方法了，只需要一个 yiled 关键字。

46.3 GBDT 为什么拟合负梯度而不是残差

函数中下降最快的方向是导数方向，同理：GBDT 中，损失函数减小最快的方向也是本身的导数方向。当损失函数为均方误差时，损失函数的负梯度和残差一样，但当损失函数不为均方误差时，本质上拟合的还是负梯度。

46.4 用 python 写一下堆排序

```
from collections import deque
def swap_param(L, i, j):
    L[i], L[j] = L[j], L[i]
    return L

def heap_adjust(L, start, end):
    temp = L[start]
    i = start
    j = 2 * i
    while j <= end:
        if (j < end) and (L[j] < L[j + 1]):
            j += 1
        if temp < L[j]:
            L[i] = L[j]
            i = j
            j = 2 * i
        else:
            break
    L[i] = temp

def heap_sort(L):
    L_length = len(L) - 1
    first_sort_count = L_length / 2
    for i in range(first_sort_count):
        heap_adjust(L, first_sort_count - i, L_length)
    for i in range(L_length - 1):
        L = swap_param(L, 1, L_length - i)
        heap_adjust(L, 1, L_length - i - 1)
    return [L[i] for i in range(1, len(L))]

def main():
    L = deque([50, 16, 30, 10, 60,  90,  2, 80, 70])
    L.appendleft(0)
    print heap_sort(L)

if __name__ == '__main__':
    main()
```

46.5 用 python 写一下快排排序

```python
def getPartitionIndex(arr, left, right):
    pivote = arr[left]
    while left < right:
        while arr[right] >= pivote and right > left:
            right -= 1
        arr[left] = arr[right]
        while arr[left] <= pivote and right > left:
            left += 1
        arr[right] = arr[left]
    arr[left] = pivote
    return left
def quickSort(arr, left=None, right=None):
    # print(arr)
    left = 0 if not isinstance(left, (int, float)) else left
    right = len(arr)-1 if not isinstance(right, (int, float)) else right
    if left < right:
        partitionIndex = getPartitionIndex(arr, left, right)
        quickSort(arr, left, partitionIndex)
        quickSort(arr, partitionIndex+1, right)
    return arr

if __name__ == '__main__':
    arr = [3, 2, 1, 5, 0, 4]
    print(quickSort(arr))
```

46.6 NLG 的评估指标有哪些

(1) BLEU (Bilingual Evaluation Understudy)

(2) ROUGE (Recall Oriented Understudy for Gisting Evaluation)

BLEU 是机器翻译中使用最广泛的评估指标，可以看成是精确率，公式如下：

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} W_n \log P_n\right)$$

ROUGE 可以看作是召回率，有以下几种：

(1)ROUGE-N：计算 n-gram 的召回率，即算出候选译文和参考译文重合的 n-gram 个数占参考译文的比例；

(2)ROUGE-L：计算最长公共子序列（LCS）的召回率；

(3)ROUGE-W：计算加权的最长公共子序列的召回率（weighted LCS）；

(4) ROUGE-S：计算 skip-bigram 的召回率（any pair of words in sentence order allowing for arbitrary gaps）。

46.7 二叉树的最大路径和

思路：递归

本质就是后序遍历，对于一个二叉树节点，计算左子树和右子树的最大路径和，再加上自己的值，得到该节点的最大路径和。

首先实现一个简化函数 maxGain(node)，计算二叉树中一个节点的最大贡献值；

计算二叉树的最大路径和，对于二叉树中的一个节点，该节点的最大路径和取决于该节点的值与该节点的左右子节点的最大贡献值，如果子节点的最大贡献值为正，则计入该节点的最大路径和，否则不计入该节点的最大路径和；

维护一个全局变量 maxSum 存储最大路径和，在递归过程中更新 maxSum 的值。

代码如下：

```
1.  class Solution:
2.      def __init__(self):
3.          self.maxSum = float("-inf")
4.      def maxPathSum(self, root: TreeNode) -> int:
5.          def maxGain(node):
6.              if not node:
7.                  return 0
8.              leftGain = max(maxGain(node.left),0)
9.              rightGain = max(maxGain(node.right),0)
10.             priceNewPath = node.val + leftGain + rightGain
11.
12.             self.maxSum = max(self.maxSum, priceNewPath)
13.
14.             return node.val + max(leftGain, rightGain)
15.
16.         maxGain(root)
17.         return self.maxSum
```

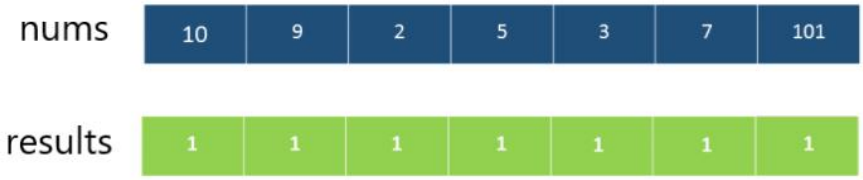
时间复杂度：O(N)

空间复杂度：O(N)

46.8 attention 为什么要除以根号下 dk

QK 进行点击之后，值之间的方差会较大，也就是大小差距会较大；如果直接通过 Softmax 操作，会导致大的更大，小的更小；进行缩放，会使参数更平滑，训练效果更好。

46.9 最长上升子序列



如上图所示，用 nums 表示原数组，results[i]表示截止到 nums[i] 当前最长递增子序列长度为 results[i]，初始值均为 1；

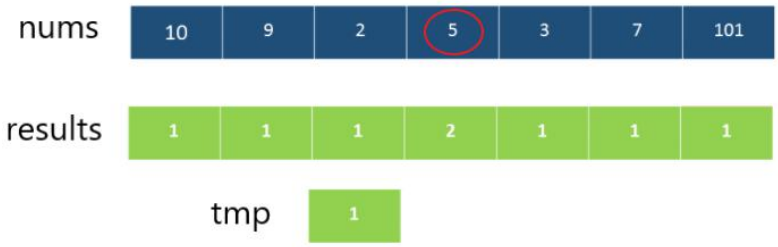
因为要找处递增序列，所以我们只需要找出 nums[i]>nums[j]（其中 j < i）的数，并将对应的 results[j]保存在 tmp 临时列表中，然后找出最长的那个序列将 nums[i]附加其后面；

示例：假设 nums[i] = 5, i = 3

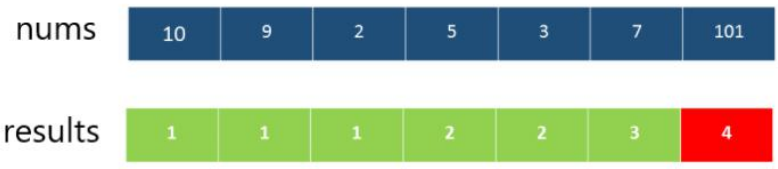
更新 results[i]时只需要考虑前面小于 results[i]的项，所以 results[0-2]均为 1；

此时 nums[i]前面小于 5 的项有 nums[2]，需要将 results[2]保存报 tmp 中，然后选取 tmp 中最大的数值并加 1 后赋值给 results[i], 即 results[5] = 2

tmp 是临时列表更新 results 之后需要清空；



根据上述流程更新所有 results[i]，即可得到下图：



此时我们只需要找出 results 中最大的那个值，即为最长递增子序列的长度。

参考代码如下：

```
1.  class Solution:
2.      def lengthOfLIS(self, nums: List[int]) -> int:
3.          # 特殊情况判断
4.          if len(nums) == 0: return 0
5.
```

```
        results = [1]*len(nums)
        for i in range(1, len(nums)):
            tmp = []
            for j in range(i):
                if nums[i] > nums[j]:
                    tmp.append(results[j])
            if tmp: results[i] = max(tmp) + 1
        return max(results)
```

46.10 神经网络有什么调参技巧?

哪些参数可以调?

(1) 网络设计相关参数：网络层数、不同层的搭建顺序、隐藏层神经元的参数设置、loss 的选择、正则化参数

(2) 训练过程相关参数：网络权重初始化方法、学习率、迭代次数、batch_size。

什么时候需要调参?

(1) 欠拟合：优化数据集(数据清洗)、增加训练迭代次数、添加更多的层，增大神经元参数等

(2) 过拟合：增加样本数量(数据增强)，加正则化参数(dropout)，使用早停机制等

第四十七篇 2021 年 10 月 25 日-京东 NLP 工程师一面 -- 10 道

47.1 如何计算文本相似度?

1、直接使用词向量做平均得到句向量，通过余弦相似度来计算

2、直接使用词向量做平均得到句向量，通过向量距离来计算

3、使用 sentenceBert 输出两个句子各自的句向量，通过余弦相似度来计算

4、使用 sentenceBert 输出两个句子各自的句向量，拼接起来，通过全连接层，再做二分类

5、使用 simCSE 输出两个句子各自的句向量，通过余弦相似度来计算。

47.2 Bert 模型的输出一般接上一个全连接层做下游的任务，是否可以用 xgboost 代替全连接层？为什么?

1、不能使用 xgboost 代替

2、不能使用任何非参数模型代替全连接层，比如以树模型为基础的模型，SVM。

3、因为 Bert 中的参数的调整是需要通过梯度反向传播来进行梯度下降来更新的，如果梯度都没有，那么如何更新参数。如果需要梯度的话，就必须上可对参数求导的模型，如果参数可求导，那么一定是有参数模型，比如逻辑回归，全链接层。

47.3 描述下 Roberta 模型和 bert 有什么不同?

1、Roberta 可以直接看成收敛后的 bert 模型

2、在更加大量的数据集上做了 Bert 预训练任务

3、取消了 NSP 任务，只关注 MLM 任务

4、使用了动态的 MASK 方式

47.4 描述下 Albert 模型和 bert 有什么不同?

1、Albert 模型相当于乞丐版的 bert 模型

2、由于 Bert 模型效果好，很多业务上都想尝试使用 bert 来做预测，但是发现 Bert 重大的缺陷就是模型太大，不好移植，且速度慢，不能达到快速响应。

3、于是尝试想看下是否可以生成一个缩小版的 Bert，用于快速响应，且效果差不多

4、Albert 首先减少了词向量的维度

5、Albert 还尝试 block 中的参数做共享

6、以上两种方法下，极大的减少了模型参数的个数，且也达到了和 Bert-Base 差不多的效果

47.5 如何减少训练好的神经网络模型的运行的时间?

1、使用使用 GPU，或者 TPU 来做运算

2、可以尝试做剪执以减少参数

3、可以尝试做 TeacherForcing 做知识蒸馏

47.6 Bert 模型适合做什么任务，不适合做什么任务?

1、Bert 模型是 Transformer 的 encoder，适合做情感分类，文本分类，意图识别，命名实体识别，句子是否相似，句子是否有被包含，半指针半标注的 SPO 三元组抽取，问答，等任务。

2、Bert 只是 Transformer 的 encoder，不能做任何生成式的任务。比如翻译，文本摘要，问题生成，等 seq2seq 任务。

47.7 你看过哪些以 Bert 为基础的模型?

1、Roberta

2、Albert

3、Bert-wwm

4、XLnet

5、DistillationBert

6、SimBert

7、SimCSE

8、UNIgram

9、PET

10、BART

等等

47.8 如果训练集的数据量太少了怎么办?

1、可以尝试做数据增强，模拟出其他的数据

2、可以尝试使用 Prompt 的思路，做小样本学习，甚至零样本学习

3、如果是分类问题，那么可以使用 PET 模型来构建 Pattern，通过完形填空的方式来间接的解决分类问题

47.9 如果业务中，训练好的模型的标签多了一个怎么办?

1、最笨的办法就是尝试重新标记好数据集，将新的标签标记到原来的数据集中，重新训练模型 2、将新的标签重新标记到数据集中，将新标签和非新标签分开。

3、首先用一个二分类模型判断样本是否属于新标签或者非新标签，如果属于新标签则结束，如果属于非新标签，则尝试用老模型继续判断样本属于哪一个老标签。

4、直接使用 PET 的方式来训练，在 MASK 的位置多增加一个 Verbalizer，然后做微调即可，完全不用更加其他的代码

47.10 BatchNorm 和 LayerNorm 的区别?

1、BatchNorm：计算每个最小批每层的平均值和方差

2、LayerNorm：独立计算每层每一个样本的均值和方差

第四十八篇：2021 年 11 月 7 日-VIVO-CV 工程师面试题 -- 3 道

48.1 简述下你对 end to end 检测器的理解?

1、从 faser-rcnn 开始解释 Blabla，原来通常用选择性搜索方法生成 proposals 不能和 cnn 一起训练，需要各自训练各自的部分。

2、此外原来的 rcnn 阶段的分类的 svm 进行的，也不能和整个网络一起训练

48.2 线性回归和逻辑回归的区别?

1、线性回归做预测，逻辑回归做分类

2、前者拟合合适的模型函数，后者预测函数的输出值

3、参数更新：最小二乘法 vs 梯度下降

4、因变量：连续性的数据，离散的 label

举例子：饮食习惯对体重的影响，如果是输入数据（性别，饮食习惯，身高，年龄等）预测重量的具体值，是用线性回归；如果预测体型，如微胖，正常等分类，用逻辑回归。

48.3 目标检测 trick

1、数据增强

2、小目标的重采样

3、根据数据集小目标的分布进行 anchor 的调整（guide-anchor,k-means）

4、多感受野：FPN,可变型卷积

5、注意力机制 orSEnet

6、HRnet 超高分辨率的 backbone

第四十九篇：2021 年 11 月 7 日-地平线视觉算法工程师 -- 4 道

49.1 Mask-rcnn 介绍一下

敲黑板：谈到 Mask-rcnn，不如说这是一道考验介绍算法的陈述题。面试官会根据你简历做的算法 or 你提到的算法（恰好他也熟悉的 Hhh）进行提问要你介绍，说明。这里不要求同学们说的多么仔细，我建议可以这样回答

1、它基于的历史：双阶段检测器 faster-rcnn+语义分割分支

2、它的最大几个 idea，让你眼前一亮或是和你的项目论文关联度比较大的创新点

(1) 解决特征图与原始图像上的 RoI 不对齐问题 ：即 Roi_align:传统的 proposals 在生成固定长度的 roi 的过程由于二次量化时造成的位置精度损失以及双线性插值法回去看 paper!!!

(2) 掩模预测和分类预测解耦：参考 Nms 的类内抑制，对于实例分割的每个类别独立地预测一个二值掩模，每个二值掩模的类别依靠网络 RoI 分类分支给出的分类预测结果。与 FCN 不同，FCN 是多分类问题（相当于 softmax）这里类似于进行了每个 class 的伯努利 0-1 分布预测（相当于 sigmoid ）

※ 这一点回答的不太好，主要书对于实例分割不够了解，欢迎大佬补充

3、后续的改进：例如 faster-rcnn→cascade→DetecoRS 的发展

49.2 L1,L2 正则化的区别

敲黑板！！！！ L1,L2 正则化也是做机器学习 or 深度必考的!

1、包括各自的功能

(why need 正则化：防止训练产生过拟合，用复杂的模型去拟合训练集时容易出现过拟合，即泛化能力不足， 用一些惩罚项约束复杂度)

2、各自怎么约束复杂度

(L1 对模型权值的绝对值之和约束，L2 的模型权值的平方和约束！)

3、区别和特点：

L1 正则化容易得到稀疏解，L2 正则化容易得到平滑解。

原因：（1）从解空间来说（2）从梯度下降来说

49.3 说一下你知道的 cv 任务里 transformer 发展的时间线

Vit，Detr，swin , Deformable

敲黑板：有做过 transformer 的同学，基本的组件要掌握的（muti-head,位置编码，编码器，解码器，FFN 等），没有的话一般不会问~因为大概 2020 才引入到 cv 的，题外话：感觉做过了 transformer，有一些不会 cv 的面试官也可以交流了 hhh 感觉 cv 面试也有挺多是 nlp,ml 方向的老师面

49.4 解释一下位置编码

主要说了 sin-cos 方式和 embeddings

追问：你认为在 CV 中，encoder 之前的位置编码能不能去除?

没有这个的话，切分 patch 的时候，只有图像的抽象特征信息而没有位置信息，感觉不利于回归任务，分类应该问题不大，然后告诉我说，目前有针对这个 positon-encoding 的简化甚至存在的必要性的讨论让我可以去看看，这个属于一个开放问题

第五十篇：2021 年 11 月 8 日-海康威视-CV 视觉工程师 -- 2 道

50.1 手撕代码题：用一个 3*3 的卷积核对一个二维数组进行平滑滤波，输出平滑之后的结果。

```
#!  usr/bin/env python
# coding: utf-8

########      自己创建一个二维卷积滤波器（3*3）      #########
import numpy as np
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('cat.jpg',0)

kernal = np.ones((3,3), np.float32)/16
# 这里用 numpy 创建一个 3*3 的单位阵，ones 表示这是单位阵，np.float32 表示数据的格式是浮点 32 型。
# 最后单位阵的每个元素再除以 9（3*3），整个滤波器表示对一个 3*3 的矩阵上的数进行平均求和。

dst = cv2.filter2D(img, -1, kernal)
# 用一个像素点周围及其本身在内的 9 个点的像素平均值代替其本身。
# 也就是以一个像素点为中心的边长为 3 的正方形区域内像素值的平均值代替原有像素值

titles = ['srcImg','convImg']
imgs = [img, dst]

# 画图进行展示
for i in range(2):
    plt.subplot(1,2,i+1)
    plt.imshow(imgs[i], 'gray')
    plt.title(titles[i])
plt.show()
#### 可从图中看出，卷积滤波的方式会弱化边缘，使得边缘不那么清晰。
#### 去噪可以理解为降低图像获取过程中的随机因素
# 从另一方面来讲，去噪使图像内一个局部区域的像素趋于相同
```

参考：(10 条消息) python+opencv 自建二维滤波器进行卷积滤波_月下花弄影-CSDN 博客

50.2 说说神经训练计算量过大，你怎么处理?

1、减少批次，减小内存消耗

2、在 Tensorflow 中，使用 tfrecord 读取数据

3、减少模型复杂度。如调整超参，减小卷积的个数，更换轻量模型

4、减少输入大小，减少计算量

5、为了减少训练时间，首先加载预训练权重，再训练

第五十一篇：2021 年 10 月中旬—字节 AI LAB NLP 算法 -- 10 道

51.1 bert 的架构是什么 目标是什么 输入包括了什么 三个 embedding 输入是怎么综合的?

1、Bert 的结构主要是 Transformer 的 encoder 部分，其中 Bert_base 有 12 层，输出维度为 768，参数量为 110M，Bert_large 有 24 层，输出维度为 1024，参数总量为 340M。

2、Bert 的目标是利用大规模无标注语料训练，获得文本包含丰富语义信息的表征。

3、Bert 的输入：token embedding，segment embedding，position embeddimg，三个向量相加作为模型的输入。

51.2 transformer 里面每一层的主要构成有哪些

Transformer 本身是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 个 Block，Encoder 端的 Block 包括两个模块，多头 self-attention 模块以及一个前馈神经网络模块；

Decoder 端的 Block 包括三个模块，多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；

需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。

51.3 Seq2seq 模型中 decode 和 encode 的差别有哪些

encoder 是对输入的序列进行编码，编码的时候不仅可以考虑当前状态，还可以考虑前后状态，在进行 decoder 的时候不能看到当前状态之后的信息，考虑的是 encoder 的 context vector 和 decoder 上一个时刻的信息。

51.4 leetcode.121. 买卖股票的最佳时机

方法一：暴力解法

对数组进行遍历，找到后一个数与前一个数的最大差值，返回。

注意遍历 j 时要从 i + 1 进行遍历。

代码如下：

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        res = 0
        for i in range(len(prices)):
            for j in range(i+1,len(prices)):
                res = max(res,prices[j] - prices[i])
```

```
            return res
```

在 leetcode 上运行上面代码会出现超出时间限制的问题。

时间复杂度：O(n^2)

空间复杂度：O(1)

方法二：

只进行一次遍历，在遍历过程中更新两个值，股票最小值和差值最大值，更新到最后即可。

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        res = 0
        minPrice = prices[0]
        for i in range(len(prices)):
            res = max(res, prices[i] - minPrice)
            minPrice = min(minPrice, prices[i])
        return res
```

时间复杂度：O(n)

空间复杂度：O(1)

51.5 求数组中所有子数组和的最大值 剑指 Offer 42

本题为剑指 offer42 题，和 Leetcode 第 53 题相同。

题目描述给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

解析：本题在面试时要求使用动态规划的方法，因此需要找到对应的状态定义（子问题）和状态转移方程（子问题之间的关系）

状态定义：dp[i]：表示以 nums[i] 结尾的连续子数组的最大和。

状态转移方程：

分两种情况考虑：dp[i-1] <= 0 和 dp[i-1] > 0

1、如果 dp[i - 1] > 0，那么可以把 nums[i] 直接接在 dp[i - 1] 表示的那个数组的后面，得到和更大的连续子数组；

2、如果 dp[i - 1] <= 0，那么 nums[i] 加上前面的数 dp[i - 1] 以后值不会变大。于是 dp[i] 「另起炉灶」，此时单独的一个 nums[i] 的值，就是 dp[i]。

得到状态转移方程如下：

$$dp[i] = \begin{cases} dp[i-1] + nums[i], & if \quad dp[i-1] > 0 \\ nums[i], & if \quad dp[i-1] \leq 0 \end{cases}$$

$$dp[i] = \max\{nums[i],\ dp[i-1] + nums[i]\}$$

整理可得：

代码如下：

```
1.  class Solution:
2.      def maxSubArray(self, nums: List[int]) -> int:
3.          size = len(nums)
4.          pre = 0
5.          res = nums[0]
6.          for i in range(size):
7.              pre = max(nums[i], pre + nums[i])
8.              res = max(res, pre)
9.          return res
```

51.6 编辑距离 leetcode 72

编辑距离 给你两个单词 word1 和 word2，请你计算出将 word1 转换成 word2 所使用的最少操作数 。

你可以对一个单词进行如下三种操作：

(1) 插入一个字符

(2) 删除一个字符

(3) 替换一个字符

示例：

输入：word1 = "horse", word2 = "ros"

输出：3

解释：

horse -> rorse (将 'h' 替换为 'r')

rorse -> rose (删除 'r')

rose -> ros (删除 'e')

解题思路：

编辑距离样例网址：https://alchemist-al.com/algorithms/edit-distance

		G	C	C	G	T
	0	1	2	3	4	5
T	1	1	① 2	3 ②	4	4
A	2	2	③ 2	3 ④	4	5
G	3	2	3	3	3	4
G	4	3	3	4	3	4
A	5	4	4	4	4	4

图中数字表示序列“TAGGA”依次更改为“GCCCGT”对应的编辑距离。

① ：表示"T" => "GC"的编辑距离为 2， 即用"G"替换"T" 再插入"C";

② ： 表示"T" => "GCC"的编辑距离为 3， 即用"G"替换"T" 再 2 次插入"C";

③ ： 表示"TA" => "GC"的编辑距离为 2， 即用"G"替换"T" 再用"C"替换"A";

由上述内容推理④ ： S1="TA" => S2="GCC"：分为如下两种情况：

- 如果 S1[-1] == S2[-1]，那么问题可以转换为 S1[:-1] => S2[-1], 即计算除最后一位之外的编辑距离，也就是④左上角位置对应的编辑距离。
- 如果 S1[-1] != S2[-1]，即图中的情况 "A" != "C"，此时对应的编辑距离为：（①，②，③）最小值 然后加 1，即为 S1 => S2 的编辑距离；
  - 如果最小值为① 表示：在 S1[:-1] => S2[:-1] 的基础上再将 S1[-1]替换为 S2[-1]；
  - 如果最小值为② 表示：在 S1[:-1] => S2 的基础上再删除 S1[-1]；
  - 如果最小值为③ 表示：在 S1 => S2[:-1] 的基础上再插入 S2[-1]；
- 第一行数字和第一列数字表示将空字符串变成相应字符所需要的编辑距离；

上述解题思路对应的代码如下：

```
1.  class Solution:
2.      def minDistance(self, word1: str, word2: str) -> int:
3.          n1 = len(word1)
4.          n2 = len(word2)
5.          # 存储当前元素的编辑距离
6.          dp = [[0] * (n2 + 1) for _ in range(n1 + 1)]
7.          # 第一行
8.          for j in range(1, n2 + 1):
9.              dp[0][j] = dp[0][j-1] + 1
10.         # 第一列
11.         for i in range(1, n1 + 1):
12.             dp[i][0] = dp[i-1][0] + 1
13.
14.         for i in range(1, n1 + 1):
```

```
            for j in range(1, n2 + 1):
                # 相等时 copy 左上角的值
                if word1[i-1] == word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                # 不相等时：mmin(上，左、左上角) + 1
                else:
                    dp[i][j] = min(dp[i][j-1], dp[i-1][j], dp[i-1][j-
                     1] ) + 1
        #print(dp)
        return dp[-1][-1]
```

51.7 bert 有什么可以改进的地方

问题：中文 BERT 是以汉字为单位训练模型的，而单个汉字是不一表达像词语或者短语具有的丰富语义。

改进：ERNIE 模型，给模型输入知识实体，在进行 mask 的时候将某一个实体的汉字全都 mask 掉，从而让模型学习知识实体。

BERT-WWM：短语级别遮盖（phrase-level masking）训练。

问题：NSP 任务会破坏原本词向量的性能。

改进：RoBERTa，移除了 NSP 任务，同时使用了动态 Mask 来代替 Bert 的静态 mask。

51.8 bert 的 mask 策略

随机 mask 每一个句子中 15%的词，用其上下文来做预测，其中，

80%的是采用[mask]，my dog is hairy → my dog is [MASK]

10%的是随机取一个词来代替 mask 的词，my dog is hairy -> my dog is apple

10%的保持不变，my dog is hairy -> my dog is hairy

51.9 Word2vec 的两种训练目标是什么 其中 skip-gram 训练的 loss function 是什么

CBOW 和 Skip-Gram 模型。

$$\log P(w_o \mid w_c) = \boldsymbol{u}_o^\top \boldsymbol{v}_c - \log \left(\sum_{i \in \mathcal{V}} \exp(\boldsymbol{u}_i^\top \boldsymbol{v}_c) \right)$$

51.10 BPE 分词了解吗?

BPE 与 Wordpiece 都是首先初始化一个小词表，再根据一定准则将不同的子词合并。词表由小变大。

BPE 与 Wordpiece 的最大区别在于，如何选择两个子词进行合并：BPE 选择频数最高的相邻子词合并，而 WordPiece 选择能够提升语言模型概率最大的相邻子词加入词表。

第五十二篇：11 月 15 日-阿里-计算机视觉工程师 -- 2 道

52.1 BN 大致的计算流程?

1、计算样本均值。

2、计算样本方差。

3、样本数据标准化处理。

4、进行平移和缩放处理。引入了γ和β两个参数。来训练γ和β两个参数。引入了这个可学习重构参数γ、β，让我们的网络可以学习恢复出原始网络所要学习的特征分布。

52.2 BN 的优点?

1、加快训练速度，这样我们就可以使用较大的学习率来训练网络。

2、提高网络的泛化能力。

3、BN 层本质上是一个归一化网络层，可以替代局部响应归一化层（LRN 层）。

4、可以打乱样本训练顺序（这样就不可能出现同一张照片被多次选择用来训练）论文中提到可以提高 1%的精度。

第五十三篇：11 月 20 日-快手 计算机视觉实习生面经 人脸人体方向 -- 5 道

53.1 stacked hourglass 和 u-net 的 skip connection 有什么区别?

他们的相似之处有如下三点：

1、结构对称，成 U 型；

2、含有 skip connection 连接，底层和高层特征有规律的融合；均是原图下采样-上采样至目标图的流程，类似沙漏(hourglass)形状。当然，他们也有不同的地方：

3、前者的单个方块是 Resenet block，后者就是简单的卷积核；前者可以做多个类似结构的堆叠，后者是单个 U 型

53.2 hrnet 的数据融合有什么特别的?

大多数现有方法从由高到低分辨率网络产生的低分辨率表示中恢复高分辨率表示。相反，本文在整个过程中保持高分辨率的表示。我们将高分辨率子网开始作为第一阶段，逐步添加高到低分辨率子网以形成更多阶段，并行连接多个子网，每个子网具有不同的分辨率。我们进行重复的多尺度融合，使得高到低分辨率表示可以重复从其他分辨率的表示获取信息，从而导致丰富的高分辨率表示。因此，预测的关键点热图可能更准确，空间更精确。

53.3 dropout 作用，训练测试有啥不一样的地方?

dropout 在训练时，以一定的概率 **p** 来 drop 掉相应的神经网络节点，以(1-p)的概率来保留相应的神经网络节点，这相当于每一次训练时模型的网络结构都不一样，也可以理解为训练时添加了不同的数据，所以能够有效减少**过拟合。**

问题呢，是出在测试时，因为训练的时候以**概率 p**drop 了一些节点，比如 dropout 设置为 0.5，隐藏层共有 6 个节点，那训练的时候有 3 个节点的值被丢弃，而测试的时候这 6 个节点都被保留下来，这就导致了**训练**和**测试**的时候以该层节点为输入的下一层的神经网络节点获取的**期望**会有量级上的差异。为了解决这个问题，在训练时对当前 dropout 层的输出数据**除以（1-p）**，之后再输入到下一层的神经元节点，以作为失活神经元的补偿，以使得在训练时和测试时每一层的输入有大致相同的期望。

53.4 1*1 卷积核作用

1、实现跨通道的交互和信息整合

2、进行卷积核通道数的降维和升维

3、对于单通道 feature map 用单核卷积即为乘以一个参数，而一般情况都是多核卷积多通道，实现多个 feature map 的线性组合

4、可以实现与全连接层等价的效果。如在 faster-rcnn 中用 1x1xm 的卷积核卷积 n（如 512）个特征图的每一个位置（像素点），其实对于每一个位置的 1x1 卷积本质上都是对该位置上 n 个通道组成的 n 维 vector 的全连接操作。

53.5 实验指标讲一下?

参考回答:

1、准确率（Accuracy）

2、精确率/查准率（Precision）

3、查全率（Recall）

4、F1-score

5、ROC 和 AUC

6、Precision-Recall 曲线

7、IOU 和 mIOU

第五十四篇：11 月 23 日 某车企 NLP 算法岗 -- 10 道

54.1 CNN 原理及优缺点

CNN 是一种前馈神经网络，通常包含 5 层，输入层，卷积层，激活层，池化层，全连接 FC 层，其中核心部分是卷积层和池化层。

优点：共享卷积核，对高维数据处理无压力；无需手动选取特征。

缺点：需要调参；需要大量样本。

54.2 word2vec 的两种优化方式

1、第一种改进为基于层序 softmax 的模型。

首先构建哈夫曼树，即以词频作为 n 个词的节点权重，不断将最小权重的节点进行合并，最终形成一棵树，权重越大的叶子结点越靠近根节点，权重越小的叶子结点离根节点越远。

然后进行哈夫曼编码，即对于除根节点外的节点，左子树编码为 1，右子树编码为 0。

最后采用二元逻辑回归方法，沿着左子树走就是负类，沿着右子树走就是正类，从训练样本中学习逻辑回归的模型参数。

优点：计算量由 V（单词总数）减小为 log2V；高频词靠近根节点，所需步数小，低频词远离根节点。

2、第二种改进为基于负采样的模型。

通过采样得到少部分的负样本，对正样本和少部分的负样本，利用二元逻辑回归模型，通过梯度上升法，来得到每个词对应的模型参数。具体负采样的方法为：根据词频进行采样，也就是词频越大的词被采到的概率也越大。

54.3 描述下 CRF 模型及应用

给定一组输入随机变量的条件下另一组输出随机变量的条件概率分布密度。条件随机场假设输出变量构成马尔科夫随机场，而我们平时看到的大多是线性链条随机场，也就是由输入对输出进行预测的判别模型。求解方法为极大似然估计或正则化的极大似然估计。

CRF 模型通常应用于优化命名实体识别任务。

54.4 CNN 的卷积公式

卷积层计算公式如下：

$$O = \frac{(W - K + 2P)}{S} + 1$$

其中，W 为输入大小，K 为卷积核大小，P 为 padding 大小，S 为步幅。

如果，想保持卷积前后的特征图大小相同，通常会设定 padding 为：

$$Padding = \frac{(K - 1)}{2}$$

54.5 逻辑回归用的什么损失函数

逻辑回归是在数据服从伯努利分布的假设下，通过极大似然的方法，运用梯度下降法来求解参数，从而达到将数据二分类的目的。

逻辑回归中使用的损失函数为对数损失，损失函数具体如下：

$$C = \frac{1}{n} \sum [y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})]$$

54.6 transformer 结构

Transformer 本身是一个典型的 encoder-decoder 模型，Encoder 端和 Decoder 端均有 6 个 Block，Encoder 端的 Block 包括两个模块，多头 self-attention 模块以及一个前馈神经网络模块；

Decoder 端的 Block 包括三个模块，多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；

需要注意：Encoder 端和 Decoder 端中的每个模块都有残差层和 Layer Normalization 层。

54.7 elmo 和 Bert 的区别

BERT 采用的是 Transformer 架构中的 Encoder 模块;

GPT 采用的是 Transformer 架构中的 Decoder 模块;

ELMo 采用的双层双向 LSTM 模块。

54.8 elmo 和 word2vec 的区别

elmo 词向量是包含上下文信息的，不是一成不变的，而是根据上下文而随时变化。

54.9 lstm 与 GRU 区别

1、LSTM 和 GRU 的性能在很多任务上不分伯仲；

2、GRU 参数更少，因此更容易收敛，但是在大数据集的情况下，LSTM 性能表现更好；

3、GRU 只有两个门（update 和 reset），LSTM 有三个门（forget，input，output），GRU 直接将 hidden state 传给下一个单元，而 LSTM 用 memory cell 把 hidden state 包装起来。

54.10 Xgboost 的分裂依据、loss 函数

xgboost 目标函数：

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

$$\text{其中，} g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

Xgboost 的分裂是根据划分前后目标函数的增益值进行决定，找到所有特征增益最大的点进行分裂。

$$\begin{aligned} Gain &= Obj_{L+R} - (Obj_L + Obj_R) \\ &= [-\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma] - [-\frac{1}{2} (\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda}) + 2\gamma] \\ &= \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \end{aligned}$$

第五十五篇：携程秋招算法二面 10 道

55.1 Jieba 分词的原理是什么

1、首先用正则表达式将中文段落粗略的分成一个个句子。

2、将每个句子构造成有向无环图，之后寻找最佳切分方案。

3、最后对于连续的单字，采用 HMM 模型将其再次划分。

55.2 特征工程的常用方法

1、对时间戳处理

2、对离散型变量进行独热编码

3、对连续型变量进行分箱/分区

4、特征缩放

5、特征选择

6、特征衍生（特征交叉）

55.3 随机森林和 GBDT 区别

随机森林采用的 bagging 思想，而 GBDT 采用的 boosting 思想。

这两种方法都是 Bootstrap 思想的应用，Bootstrap 是一种有放回的抽样方法思想。

虽然都是有放回的抽样，**但二者的区别在于：**Bagging 采用有放回的均匀取样，而 Boosting 根据错误率来取样（Boosting 初始化时对每一个训练样例赋相等的权重 1 / n，然后用该算法对训练集训练 t 轮，每次训练后，对训练失败的样例赋以较大的权重），因此 Boosting 的分类精度要优于 Bagging。

Bagging 的训练集的选择是随机的，各训练集之间相互独立，弱分类器可并行，而 Boosting 的训练集的选择与前一轮的学习结果有关，是串行的。

组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只能由回归树组成。

组成随机森林的树可以并行生成；而 GBDT 只能是串行生成。

对于最终的输出结果而言，随机森林采用多数投票等；而 GBDT 则是将所有结果累加起来，或者加权累加起来。

随机森林对异常值不敏感；GBDT 对异常值非常敏感。

随机森林对训练集一视同仁；GBDT 是基于权值的弱分类器的集成。

随机森林是通过减少模型方差提高性能；GBDT 是通过减少模型偏差提高性能。

55.4 xgboost 和 lightgbm 区别

1、内存更小

XGBoost 使用预排序后需要记录特征值及其对应样本的统计值的索引，而 LightGBM 使用了直方图算法将特征值转变为 bin 值，且不需要记录特征到样本的索引，将空间复杂度大大降低，极大的减少了内存消耗；

LightGBM 采用了直方图算法将存储特征值转变为存储 bin 值，降低了内存消耗；LightGBM 在训练过程中采用互斥特征捆绑算法减少了特征数量，降低了内存消耗。

2、速度更快

LightGBM 采用了直方图算法将遍历样本转变为遍历直方图，极大的降低了时间复杂度；LightGBM 在训练过程中采用单边梯度算法过滤掉梯度小的样本，减少了大量的计算；LightGBM 采用了基于 Leaf-wise 算法的增长策略构建树，减少了很多不必要的计算量；LightGBM 采用优化后的特征并行、数据并行方法加速计算，当数据量非常大的时候还可以采用投票并行的策略；LightGBM 对缓存也进行了优化，增加了 Cache hit 的命中率。

55.5 LSTM 的结构

包括遗忘门，输入门和输出门三种。

GRU 与 LSTM 区别

（1）LSTM 和 GRU 的性能在很多任务上不分伯仲；

（2）GRU 参数更少，因此更容易收敛，但是在大数据集的情况下，LSTM 性能表现更好；

（3）GRU 只有两个门（update 和 reset），LSTM 有三个门（forget，input，output），GRU 直接将 hidden state 传给下一个单元，而 LSTM 用 memory cell 把 hidden state 包装起来。

遗忘门

55.6 RNN BPTT 的机制

BPTT（back-propagation through time）算法是常用的训练 RNN 的方法，其实本质还是 BP 算法，只不过 RNN 处理时间序列数据，所以要基于时间反向传播，故叫随时间反向传播。

BPTT 的中心思想和 BP 算法相同，沿着需要优化的参数的负梯度方向不断寻找更优的点直至收敛。综上所述，BPTT 算法本质还是 BP 算法，BP 算法本质还是梯度下降法。

55.7 RNN 为什么会出现梯度消失或者梯度爆炸

在 RNN 中经常遇到梯度消失和爆炸现象的原因：很难捕捉到长期的依赖关系，因为乘法梯度可以随着层的数量呈指数递减/递增。

55.8 sigmoid 和 softmax 的区别

Softmax 函数是二分类函数 Sigmoid 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。最大的区别在于 softmax 的计算的是一个比重，而 sigmoid 只是对每一个输出值进行非线性化。

55.9 描述下 DeepFM 模型

在处理 CTR 预估问题中，传统的方法有一个共同的缺点：对于低阶的组合特征，学习到的比较少；

但是低阶特征对于 CTR 也非常重要，于是 Google 为了同时学习低阶和高阶组合特征，提出了 Wide&Deep 模型：混合了一个 线性模型（Wide part）和 Deep 模型 (Deep part)；

这两部分模型需要不同的输入，其中 Wide part 部分的输入仍然依赖人工特征工程；

此时模型存在两个问题：

1、偏向于提取低阶或者高阶的组合特征，不能同时提取这两种类型的特征；

2、需要专业的领域知识来做特征工程；

DeepFM 在 Wide&Deep 的基础上进行改进，成功解决了上述这两个问题，并做了一些优化；

优点如下：

1、不需要预训练 FM 得到隐向量；

2、不需要人工特征工程；

3、能同时学习低阶和高阶的组合特征；

FM 模块和 Deep 模块共享 Feature Embedding 部分，可以更快、更精确的训练；

55.10 训练集、测试集和验证集的作用，训练的时候为什么要进行 shuffle

首先用训练集训练出模型，然后用验证集验证模型（注意：这是一个中间过程，此时最好的模型还未选定），根据情况不断调整模型，选出其中最好的模型（验证误差用于指导我们选择哪个模型），记录最好的模型的各项设置，然后据此再用（训练集+验证集）数据训练出一个新模型，作为最终的模型，最后用测试集评估最终的模型。

进行 shuffle：打乱数据之间的顺序，让数据随机化，避免过拟合

第五十六篇：SHAREit(茄子)算法工程师 8 道

56.1 模型有过拟合的现象，过拟合怎么办?

1、降低模型复杂度

2、增加更多的训练数据：使用更大的数据集训练模型

3、数据增强

4、正则化：L1、L2、添加 BN 层

5、添加 Dropout 策略

6、Early Stopping

7、重新清洗数据：把明显异常的数据剔除

8、使用集成学习方法：把多个模型集成在一起，降低单个模型的过拟合风险

56.2 L1 正则和 L2 正则有啥区别?

1、L1 是模型各个参数的绝对值之和。L2 是模型各个参数的平方和的开方值。

2、L1 会趋向于产生少量的特征，而其他的特征都是 0。

因为最优的参数值很大概率出现在坐标轴上，这样就会导致某一维的权重为 0 ，产生稀疏权重矩阵

L2 会选择更多的特征，这些特征都会接近于 0。

最优的参数值很小概率出现在坐标轴上，因此每一维的参数都不会是 0。当最小化||w||时，就会使每一项趋近于 0。

L1 的作用是为了矩阵稀疏化。假设的是模型的参数取值满足拉普拉斯分布。

L2 的作用是为了使模型更平滑，得到更好的泛化能力。假设的是参数是满足高斯分布。

56.3 dropout 介绍一下，训练测试有啥不一样的地方?

dropout 在训练时，以一定的概率 p 来 drop 掉相应的神经网络节点，以(1-p)的概率来保留相应的神经网络节点，这相当于每一次训练时模型的网络结构都不一样，也可以理解为训练时添加了不同的数据，所以能够有效减少过拟合。

问题呢，是出在测试时，因为训练的时候以概率 p drop 了一些节点，比如 dropout 设置为 0.5，隐藏层共有 6 个节点，那训练的时候有 3 个节点的值被丢弃，而测试的时候这 6 个节点都被保留下来，这就导致了训练和测试的时候以该层节点为输入的下一层的神经网络节点获取的期望会有量级上的差异。

为了解决这个问题，在训练时对当前 dropout 层的输出数据除以（1-p），之后再输入到下一层的神经元节点，以作为失活神经元的补偿，以使得在训练时和测试时每一层的输入有大致相同的期望。

56.4 GBDT 与 Xgboost 的区别

1、传统的 GBDT 以 CART 树作为基学习器，XGBoost 还支持线性分类器，这个时候 XGBoost 相当于 L1 和 L2 正则化的逻辑斯蒂回归（分类）或者线性回归（回归）；

2、传统的 GBDT 在优化的时候只用到一阶导数信息，XGBoost 则对代价函数进行了二阶泰勒展开，得到一阶和二阶导数；

3、XGBoost 在代价函数中加入了正则项，用于控制模型的复杂度。从权衡方差偏差来看，它降低了模型的方差，使学习出来的模型更加简单，放置过拟合，这也是 XGBoost 优于传统 GBDT 的一个特性；

4、shrinkage（缩减），相当于学习速率（XGBoost 中的 eta）。XGBoost 在进行完一次迭代时，会将叶子节点的权值乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。（GBDT 也有学习速率）；

5、列抽样：XGBoost 借鉴了随机森林的做法, 支持列抽样, 不仅防止过 拟合，还能减少计算；

6、对缺失值的处理: 对于特征的值有缺失的样本，XGBoost 还可以自动 学习出它的分裂方向；

7、XGBoost 工具支持并行。Boosting 不是一种串行的结构吗?怎么并行 的？注意 XGBoost 的并行不是 tree 粒度的并行，XGBoost 也是一次迭代完才能进行下一次迭代的（第 t 次迭代的代价函数里包含了前面 t-1 次迭代的预测值）。XGBoost 的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），XGBoost 在训练之前，预先对数据进行了排序，然后保存为 block 结构，后面的迭代 中重复地使用这个结构，大大减小计算量。这个 block 结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

56.5 LightGBM 与 XGBoost 的区别

1、xgboost 采用的是 level-wise 的分裂策略，而 lightGBM 采用了 leaf-wise 的策略，区别是 xgboost 对每一层所有节点做无差别分裂，可能有些节点的增益非常小，对结果影响不大，但是 xgboost 也进行了分裂，带来了不必要的开销。leaft-wise 的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂，如此递归进行，很明显 leaf-wise 这种做法容易过拟合，因为容易陷入比较高的深度中，因此需要对最大深度做限制，从而避免过拟合。

2、lightgbm 使用了基于 histogram 的决策树算法，这一点不同与 xgboost 中的 exact 算法，histogram 算法在内存和计算代价上都有不小优势。

（1）内存：直方图算法的内存消耗为(#data* #features * 1Bytes)(因为对特征分桶后只需保存特征离散化之后的值)，而 xgboost 的 exact 算法内存消耗为：(2 * #data * #features* 4Bytes)，因为 xgboost 既要保存原始 feature 的值，也要保存这个值的顺序索引，这些值需要 32 位的浮点数来保存。

（2）计算：预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值，时间为(#data),而直方图算法只需要遍历桶就行了，时间为(#bin)

56.6 特征工程怎么做的?

1、数据预处理：处理缺失值、图片数据扩充、处理异常值、处理类别不平衡问题

2、特征缩放：归一化、正则化

3、特征编码：序号编码(Ordinal Encoding)、独热编码(One-hot Encoding)、二进制编码(Binary Encoding)

离散化

1、特征选择：过滤式(filter)、包裹式(wrapper)、嵌入式(embedding)

2、特征提取：降维、图像特征提取、文本特征提取

56.7 1*1 卷积核作用

1、实现跨通道的交互和信息整合

2、进行卷积核通道数的降维和升维

3、对于单通道 feature map 用单核卷积即为乘以一个参数，而一般情况都是多核卷积多通道，实现多个 feature map 的线性组合

4、可以实现与全连接层等价的效果。如在 faster-rcnn 中用 1x1xm 的卷积核卷积 n（如 512）个特征图的每一个位置（像素点），其实对于每一个位置的 1x1 卷积本质上都是对该位置上 n 个通道组成的 n 维 vector 的全连接操作。

56.8 常用评估指标介绍下

1、准确率（Accuracy）

定义：

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{.}$$

即所有分类正确的样本占全部样本的比例

2、精确率/查准率（Precision）

定义：

$$Precision = \frac{TP}{TP + FP} \tag{.}$$

即预测是正例的结果中，确实是正例的比例。Precision 同样是衡量误检

3、查全率（Recall）

定义：

$$Recall = \frac{TP}{TP + FN} \tag{.}$$

即所有正例的样本中，被找出的比例。Recall 同样是衡量漏检

4、F1-score

定义：

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{.}$$

衡量 Precision 和 Recall 之间的联系。

5、ROC 和 AUC

AUC 是一种模型分类指标，且仅仅是二分类模型的评价指标。AUC 是 Area Under Curve 的简称，那么 Curve 就是 ROC（Receiver Operating Characteristic），翻译为"接受者操作特性曲线"。也就是说 ROC 是一条曲线，AUC 是 一个面积值。

6、Precision-Recall 曲线

PR 曲线的横坐标是精确率 P，纵坐标是召回率 R。

此时曲线上的点就对应 F1。P-R 曲线同样可以用 AUC 衡量，AUC 大的曲线越好。

7、IOU 和 mIOU

$$IOU = Jaccard = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{.}$$

就是 IOU（Intersection over Union），交集占并集的大小。

公式其实很简单，就是交集占并集的大小。

mIOU 一般都是基于类进行计算的，将每一类的 IOU 计算之后累加，再进行平均，得到的就是 mIOU。

8、AP 和 mAP

AP（average Precision）和 mAP（mean average Precision）常用于目标检测任务中。AP 就是每一类的 Precision 的平均值。而 mAP 是所有类的 AP 的平均值。

第五十七篇：百度实习算法岗面试题

57.1 LR 推导目标函数并求梯度

逻辑回归损失函数及梯度推导公式如下：

$$
\begin{aligned}
L(w) &= \sum_i \left(y_i * \operatorname{logh}\left(x_i\right) + \left(1-y_i\right) * \log\left(1-h\left(x_i\right)\right)\right) \\
&= \sum_i y_i \left(\operatorname{logh}\left(x_i\right) - \log\left(1-h\left(x_i\right)\right)\right) + \log\left(1-h\left(x_i\right)\right) \\
&= \sum_i y_i \log \frac{h\left(x_i\right)}{1-h\left(x_i\right)} + \log\left(1-h\left(x_i\right)\right) \\
&= \sum_i y_i \left(w^T x_i\right) + \log\left(1 - \frac{1}{1+e^{-wx_i}}\right) \\
&= \sum_i \left(y_i * \left(w^T x_i\right) - \log\left(1+e^{w^T x_i}\right)\right)
\end{aligned}
$$

求导：

$$
\begin{aligned}
\frac{dL}{dw} &= yx - \frac{1}{1+e^{w^T x}} * e^{w^T x} * x \\
&= x(y - h(x))
\end{aligned}
$$

57.2 GBDT 和 XGBOOST 差别

利用二阶信息；

处理缺失值；

弱分类器选择；

列抽样和行抽样；

正则项做预剪枝；

并行化处理（特征排序等）。

57.3 Batch Normalization 缺点

batch 太小,会造成波动大；对于文本数据，不同有效长度问题；测试集上两个数据均值和方差差别很大就不合适了

附：LN 是对一个样本的一个时间步上的数据进行减均除标准差，然后再回放（参数学习）对应到普通线性回归就是一层节点求均除标准差。

57.4 分词如何做

基于规则（超大词表）；基于统计（两字同时出现越多，就越可能是词）；基于网络 LSTM+CRF 词性标注，也可以分词。

57.5 Adam 缺点

后期梯度很小，几乎不动了，没有 SGD 好，前期快是优点；泛化能力不强。

57.6 各类激活函数优缺点

常见的激活函数有：Sigmoid、Tanh、ReLU、Leaky ReLU

Sigmoid 函数：

特点：

它能够把输入的连续实值变换为 0 和 1 之间的输出，特别的，如果是非常大的负数，那么输出就是 0；如果是非常大的正数，输出就是 1。

缺点：

缺点 1：在深度神经网络中梯度反向传递时导致梯度消失，其中梯度爆炸发生的概率非常小，而梯度消失发生的概率比较大。

缺点 2：Sigmoid 的 output 不是 0 均值（即 zero-centered）。

缺点 3：其解析式中含有幂运算，计算机求解时相对来讲比较耗时。对于规模比较大的深度网络，这会较大地增加训练时间。

Tanh 函数：

特点：它解决了 Sigmoid 函数的不是 zero-centered 输出问题，收敛速度比 sigmoid 要快，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。

ReLU 函数：

特点：

1、ReLu 函数是利用阈值来进行因变量的输出，因此其计算复杂度会比剩下两个函数低（后两个函数都是进行指数运算）

2、ReLu 函数的非饱和性可以有效地解决梯度消失的问题，提供相对宽的激活边界。

3、ReLU 的单侧抑制提供了网络的稀疏表达能力。

ReLU 的局限性:在于其训练过程中会导致神经元死亡的问题。

这是由于函数 f(x)=max(0,x)导致负梯度在经过该 ReLU 单元时被置为 0，且在之后也不被任何数据激活，即流经该神经元的梯度永远为 0，不对任何数据产生响应。在实际训练中，如果学习率（Learning Rate）设置较大，会导致超过一定比例的神经元不可逆死亡，进而参数梯度无法更新，整个训练过程失败。

Leaky ReLu 函数：

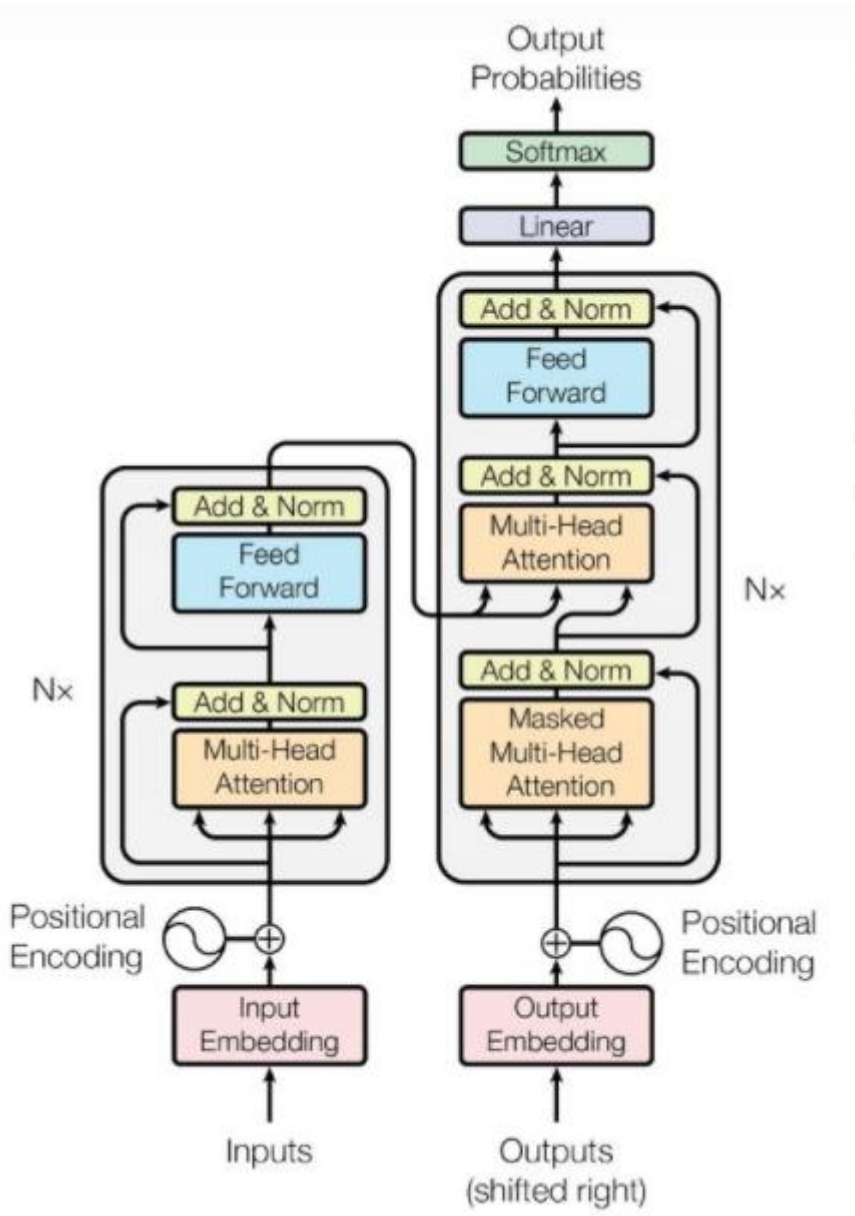
LReLU 与 ReLU 的区别在于， 当 z<0 时其值不为 0，而是一个斜率为 a 的线性函数，一般 a 为一个很小的正常数， 这样既实现了单侧抑制，又保留了部分负梯度信息以致不完全丢失。但另一方面，a 值的选择增加了问题难度，需要较强的人工先验或多次重复训练以确定合适的参数值。

基于此，参数化的 PReLU（Parametric ReLU）应运而生。它与 LReLU 的主要区别是将负轴部分斜率 a 作为网络中一个可学习的参数，进行反向传播训练，与其他含参数网络层联合优化。而另一个 LReLU 的变种增加了 “随机化” 机制，具体地，在训练过程中，斜率 a 作为一个满足某种分布的随机采样；测试时再固定下来。Random ReLU（RReLU）在一定程度上能起到正则化的作用。

ELU 函数：

ELU 函数是针对 ReLU 函数的一个改进型，相比于 ReLU 函数，在输入为负数的情况下，是有一定的输出的，而且这部分输出还具有一定的抗干扰能力。这样可以消除 ReLU 死掉的问题，不过还是有梯度饱和和指数运算的问题。

57.7 画一下 Transformer 结构图



57.8 word2vector 负采样时为什么要对频率做 3/4 次方?

在保证高频词容易被抽到的大方向下，通过权重 3/4 次幂的方式，适当提升低频词、罕见词被抽到的概率。如果不这么做，低频词，罕见词很难被抽到，以至于不被更新到对应 Embedding。

第五十八篇：bigo 推荐算法一/二面 面经（9 月中旬）

58.1 lstm 原理三个门作用和 sigmoid 函数 tanh 使用，梯度消失问题如何解决，rnn 为什么不能，缺点如何造成的。lstm 如何解决长期记忆问题 。

LSTM 是循环神经网络 RNN 的变种，包含三个门，分别是输入门，遗忘门和输出门。

sigmoid 函数主要是决定什么值需要更新；

tanh 函数：创建一个新的候选值向量，生成候选记忆。

rnn 梯度消失的原因：很难捕捉到长期的依赖关系，因为乘法梯度可以随着层的数量呈指数递减/递增。

LSTM 中 ct 到 ct-1 的路径上梯度不会消失，并不能保证其他路径上梯度不会消失。LSTM 可以缓解梯度消失，并不能消除，所以其可以解决 RNN 长期依赖的问题。

58.2 bert 原理和注意力机制介绍一下。

bert 原理：bert 是基于 Transformer encoder 的双向编码器，模型输入包含 token embedding、position embedding、segment embedding；输出包含 CLS 及每个字对应的向量表示，预训练任务为 MLM 与 NSP。

注意力机制就是对输入权重分配的关注，最开始使用到注意力机制是在编码器-解码器(encoder-decoder)中, 注意力机制通过对编码器所有时间步的隐藏状态做加权平均来得到下一层的输入变量。

58.3 召回和排序中有哪些模型，原理都了解吗?

召回模型：Youtube DNN、DSSM 双塔模型、MIND 用户多兴趣网络

排序模型：FM、Wide&Deep Model，DeepFM, Deep&Cross、DIN、DIEN、DSIN、BST 等等。

58.4 lr 特征为什么要离散化

数据角度：离散化的特征对异常数据有很强的鲁棒性；离散化特征利于进行特征交叉。**模型角度**：当数据增加/减少时，利于模型快速迭代；离散化相当于为模型引入非线性表达；离散化特征简化了模型输入，降低过拟合风险；LR 中离散化特征很容易根据权重找出 bad case。计算角度：稀疏向量内积计算速度快。（在计算稀疏矩阵内积时，可以根据当前值是否为 0 来直接输出 0 值，这相对于乘法计算是快很多的。）

58.5 auc 公式是什么，如何一句话解释 auc 的含义，数据不平衡对 auc 有影响吗，还有什么指标可以针对不平衡数据。

$$AUC = \frac{\sum_{ie\ positiveClass} rank_i - \frac{M(1+M)}{2}}{M \times N}$$

auc 的直观含义是任意取一个正样本和负样本，正样本得分大于负样本的概率。

数据不平衡对 auc 影响不大。

对于不平衡数据还可以使用 PR(Precision-Recall 曲线)。

58.6 代码题：最长不重复子串

该题为 leetcode 第 3 题。

方法：双指针 + sliding window

定义两个指针 start 和 end 得到 sliding window

start 初始为 0，用 end 线性遍历每个字符，用 recod 记录下每个字母最新出现的下标

两种情况：一种是新字符没有在 record 中出现过，表示没有重复，一种是新字符 char 在 record 中出现过，说明 start 需要更新，取 start 和 record[char]+1 中的最大值作为新的 start。

需要注意的是：两种情况都要对 record 进行更新，因为是新字符没在 record 出现过的时候需要添加到 record 中，而对于出现过的情况，也需要把 record 中对应的 value 值更新为新的下标。

代码如下：

```
1.  class Solution:
2.      def lengthOfLongestSubstring(self, s: str) -> int:
3.          record = {}
4.          start,res = 0,0
5.          for end in range(len(s)):
6.              if s[end] in record:
7.                  start = max(start, record[s[end]] + 1)
8.              record[s[end]] = end
9.              res = max(res, end - start + 1)
10.         return res
```

时间复杂度：O(n)

空间复杂度：O(1)

58.7 针对不平衡样本问题，在通过上采样后，正例：负例的比值由 1：10 变为 2：10，则 dnn 模型预测的概率会增大多少，若想要当前输出概率和原始期望想同，则需要如何操作 。

会从 1/11 增大到 1/6。

需要除以对应的增长倍数，来保证期望值不变。类似于 dropout 的使用方法，在使用 dropout 后需要在输出下一层之前除以这个 dropout。

58.8 dnn 如何评估特征有效性和重要性，其他机器学习模型呢?

训练后使用 OOB（Out of Bag）数据计算第二种方式是训练好模型之后，用 Out of Bag（或称 Test）数据进行特征重要性的量化计算。具体来说，先用训练好的模型对 OOB 数据进行打分，计算出 AUC 或其他业务定义的评估指标；接着对 OOB 数据中的每个特征：

（1）随机 shuffle 当前特征的取值；

（2）重新对当前数据进行打分，计算评估指标；

（3）计算指标变化率按照上面方式，对每个特征都会得到一个变化率，最后按照变化率排序来量化特征重要性。

参考代码如下：

```
for key in COLUMNS:
    copy = origin.copy()
    copy[key] = copy[key].sample(frac=1, random_state=1).reset_index()[key]
    cp_out = predict(copy)
    out[key] = out['tag'] - cp_out['tag']
    out[key] = out[key]**2
    print('key = ', key ,' affect = ', out[key].sum()**0.5)

pd.DataFrame(out.sum(axis=0))**0.5).sort_values(by=0 , ascending=False)
```

机器学习模型：如 Xgboost 模型

训练过程中计算训练过程中通过记录特征的分裂总次数、总/平均信息增益来对特征重要性进行量化。例如实际工程中我们会用特征在整个 GBDT、XgBoost 里面被使用的次数或者带来的总/平均信息增益来给特征重要度打分，最后进行排序。由于本身 Ensemble 模型在选择特征分裂时带有一定随机性，一般会跑多个模型然后把特征重要性求平均后排序。