24.2记录

layout: post title: 24年2月日报 subtitle: 搞论文的一个月 tags: [2024日报] comments: true author: zyk cov...

---

layout: post
title: 24年2月日报
subtitle: 搞论文的一个月
tags: [2024日报]
comments: true
author: zyk
cover-img: /assets/img/path.jpg
thumbnail-img: /assets/img/duck.jpg
share-img: /assets/img/path.jpg

2.1

1.普通bfs适用于无权图或等权图的最短路搜索，使用队列
0-1bfs针对边权仅为0或者1的图，使用双端队列，权重为0的节点会被加到队列前端被优先处理，此时注意把step也和状态存一下但不存在visited中，普通bfs可以在外面用step是因为每次都一并处理队列中所

有的状态

2.b树与b+树：

B+树看这一篇就够了（B+树查找、插入、删除全上）

基本概念为了实现动态多层索引，通常采用 B-树 和 B+树。但是，用于索引的 B-树 存在缺陷，它的...

知乎专栏

B+树相比于B树可以减少IO次数的原因：
1）B+树的非叶子节点仅存储健和子节点的指针，不存储指向数据的指针，使得每个节点能容纳更多的健，因此B+树比B树更浅，从而需要的层级跳转更少，IO次数也就更少
2）B+树中所有叶子结点串成一个链表，使得对数据的范围查询更高效，因为可以直接在链表上顺序读取，不需要进行树的遍历

3.snipaste贴图按空格进入编辑页面

4.linux中touch命令：
touch filename 如果filename指定的文件不存在，会创建一个空的文件；如果文件已经存在，touch会更新该文件的访问和修改时间

2.12

engineering coding：

1. sklearn.preprocessing 的LabelEncoder()类将类别型特征映射到不同数字，从0开始
2. ndarray以及tensor支持[i,j]这样的索引，比如Q-table[s,a]，而二维的list只能Q-table[s] [a]
3. f-string格式：f"{number:.2f}"
4. ndarray只能在内存，即'cpu'类型，而torch.tensor在创建时候为'cpu'类型，to(device)后在显存中
5. 网络包括输入层、隐藏层、输出层
6. 当ndarray、tensor当中只有一个元素时（例如1或者[1]或者[[1]]等等）才能通过.item()，得到一个标量值
7. ndarray与tensor都是列方向dim/axis为0，行方向dim/axis为1
8. 神经网络的forward中只允许用F中或者**init**定义好的层
9. list->tensor：（1）list中如果不含ndarray，使用list->tensor（2）list中如果有ndarray，使用list->ndarray->tensor
10. 假设有模型A与B，需要将A的输出作为B的输入，但训练时只训练B，可以这样做：input_B=output_A.detach()，可以将两个计算图的梯度传递断开，input_B.requires_grad=False
11. 设置seed=不是在这一次运行中生成的随机数相同，而是下一次运行生成的随机数序列与上一次运行时相同
12. 什么需要to(device)，定义的网络、网络输入、需要和网络输出运算的tensor
13. 如果网络模型参数越多模型越复杂，则gpu明显更快，并且相比参数量的增长耗时增长较慢。如果模型简单，则cpu更快，但是随着参数增多，耗时急剧增加
14. 网络中不要*=、+=这样简写，在loss.backward()时会报错，影响梯度的回传

leetcode:

1. collections.Deque为双端队列，即可以从两侧进行append以及pop，所以其既可以表示队列也可以表示栈,时间复杂度O(1)，insert(index,item)时间复杂度O(n)，remove(item)时间复杂度O(n)，支持in运算符O(n)
2. from sortedcontainers import SortedList 其插入元素与删除元素的时间复杂度都是O(logn)，而普通的list的插入与删除都是O(n)，既然这样可以直接取代堆，并且比堆更牛。sl自带的二分查找sl.bisect_right()比bisect.bisect_right(sl)快很多
3. 判断一个数是否为2的幂 num&(num-1)==0则是2的幂
4. float类型的数向下取整int(num)，向上取整math.ceil(num)
5. bisect_left与bisect_right可以限定返回的index的范围bisect_left(nums,x,begin_index,end_index(inclusive))
6. 一个set可以与一个二进制数对应，某一位为0说明该位对应的数字不在set中，为1说明对应的数字在set中
7. 子数组(子串)问题主要有两种思路：（1）双指针，使用条件需要满足两段性，即一侧满足条件，另一侧不满足（2）前缀和+哈希表，常用于统计子数组个数，不需要有两段性
8. list类型不能hash，不能放到哈希表中，可以考虑将list转为tuple或者字符串放进哈希表

9. 两个非重叠的最大值问题，可以考虑枚举分割点，或者枚举右边的选择，加左边能选择的最大值
10. 一个数组的中位数到这个数组所有值的距离之和最小
11. 根据到达时间或者先后顺序模拟通常考虑同向双指针模拟时间
12. s.split()与s.split(' ')不同，前者如果两个单词之间有多个空格或者前后面有多余空格也会去除，而后者则会分割出''
13. 最短路算法的分类，都适用于有向图以及无向图：
    （1）单源最短路
        所有边权都是正数
            使用贪心策略优化后的广度优先搜索
            朴素的Dijkstra算法O(n^2)适合稠密图
            堆优化的Dijkstra算法O(mlogn)m是图中节点的个数，适合稀疏图
        存在负权边
            Bellman-Ford O(nm)
            SPFA 一般O(m)，最坏O(nm)
    （2）多源最短路
            Floyd算法 O(n^3)
14. 存储图的方式有两种（1）邻接表我习惯用于边没有权重/边没有属性的图，用list存储每个节点的邻居（2）邻接矩阵用一个n*n矩阵存储边的权重，float('inf')代表无边
15. 四舍五入可以将该数+0.5后int()
16. acm模式下，使用sys.exit()退出程序，避免多个if else 嵌套
17. python别的进制转十进制int(x,base) x为字符串或者数字 base表示进制数
18. python中负数可以%mod，负数自动加mod
19. 求一个很长的数字除k的余数，可以通过（前n-1位数字的余数*10+int(最后一位)）%k得到
20. 位运算的左移<< 右移>> 比乘法除法快非常多

2.14

1.

📖统计学知识科普

假设检验

ab实验是一种经典的抽样研究方法，其结论要放进假设检验中考核一下。

- H0原假设：ab两组无差异
- H1备择假设：有差异

> 📌 我们希望证明H1， 拒绝H0。

- 一类错误α：弃真，即明明无差异，却被我们判别为有差异的概率。 一般控制在5%。 即置信度为95%。
- 二类错误β：取伪，即明明有差异，却被我们判别为无差异的概率。一般控制在20%。即统计功效为80%

> 📌 **上述标准业界通用，一般不会改变**，作为常数参数参与各统计结论的计算。
>
> 从一二类错误的比例设置上可以看出，人们更能接受错过一个好人（二类错误接受20%），但更忌讳认可一个坏人（一类错误接受5%）。 这在我们的业务逻辑是说得通的！**改版迭代是有各种业务成本/风险的！所以基本思路是宁缺毋滥，确保每一步走在正确的路上。**

置信区间

置信度区间是用来对一组实验数据的总体参数进行估计的区间范围。

> 举个例子，我们现在开了一个实验来优化首页arpu，其中采用了新策略B的实验组，arpu均值为2.1元，相比对照组a的arpu2元，提升了5%，置信区间为[-3%，13%]。
>
> 怎么理解此处的置信区间呢?
>
> 由于在A/B实验中我们采取小流量抽样的方式，样本不能完全代表总体，那么实际上策略B如果在总体流量中生效，不见得会获得5%的增长。真实增长率会介于[-3%，13%]之间。

值得注意的是，实际增长率并不是100%概率落在这一区间里，在计算置信区间的过程中，我们会先取一个置信水平，计算这一置信水平下的置信区间是多少，A/B实验中我们通常计算95%置信度下的置信区间。回到刚刚的例子，我们就可以得知，arpu增长率的真实取值有95%的可能落在[-3%，13%]之间。而这代表，**我们以为的+5%增长其实是不显著的**，实验效果也有可能是-3%哦!

- **规则：置信区间上下界同正或同负，则代表显著。一正一负跨过0则代表不显著。**

z检验下的计算公式如下。p1为2.1元，p2为2元。在一类错误给定的情况下（下式Z值给定），我们的置信区间大小完全依赖标准误差大小（即下式根号包着的），是一个方差/样本量的比值。**方差越大（指标天然波动越大），则区间越大越容易不显著；样本量越大，区间越小越容易显著。**

$$(\rho_1 - \rho_2) \pm z_{\frac{\alpha}{2}} \cdot \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

2.18

1.子序列DP思考套路：

（1）0-1背包 选的子序列的相邻元素是无关的

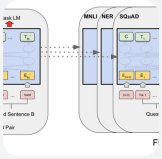
（2）LIS 选的子序列的相邻元素是相关的

2.22

1.记忆化搜索的函数return不要是list这种可变类型，因为return的是引用而不是副本，如果在外部该list被修改了，那么缓存中的对象也会跟着改变，需要return tuple(list)这种不可变类型

2.23

1.Bert模型总结：



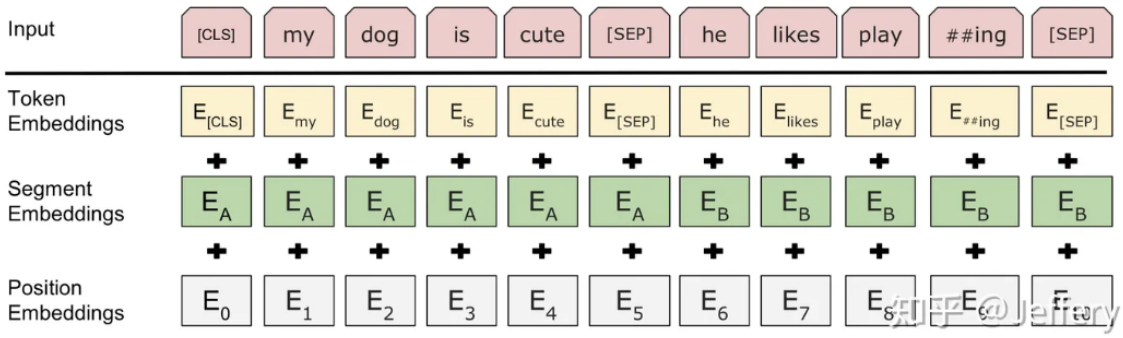
什么是BERT?

BERT的全称为Bidirectional Encoder Representation from Transformers，是一个预训练的语言表征...
知乎专栏

bert整体是transformer中的encoder部分，如果要做翻译任务，需要再接一个decoder

bert的输入的每部分中包含三部分：

> 上面提到了BERT的输入为每一个token对应的表征，实际上该表征是由三部分组成的，分别是对应的**token**，**分割**和**位置** embeddings（*位置embeddings的详细解释可参见Attention Is All You Need 或 The Illustrated Transformer*），如下图：



token表征的组成

bert的预训练任务：
（1）Masked Language Model（MLM）
在transformer编码器基础上，进一步增强模型利用上下文信息的能力
（2）Next Sentence Prediction（NSP）

这两个任务所需数据都可以从无标签的文本数据中构建（自监督性质）

2.对比学习总结：

对比学习（Contrastive Learning）综述

A.引入A.引入深度学习的成功往往依赖于海量数据的支持，其中对于数据的标记与否，可以分为监督学...
知乎专栏

对比学习主要是通过引入一个额外loss（对比损失），来辅助训练特征提取器
主要loss函数有以下几种：

（1）Contrastive Loss（狭义对比损失，因为广义下这些loss都算对比损失）
输入就是正样本对或者负样本对，没有锚点概念

$$L = \sum_{(i,j)\in P}[||f(i) - f(j)||_2^2] + \sum_{(i,k)\in N}[\max(0, m - ||f(i) - f(k)||_2)^2]$$

这里，$f(i)$ 和 $f(j)$ 分别是样本 i 和 j 经过模型映射后的特征表示，P 是正样本对集合，N 是负样本对集合，m 是一个预设的边界值，用于控制负样本对之间的最小距离。

（2）Triplet Loss （常见）
输入是一个锚点样本、一个正样本、一个负样本

$$L = \sum_{(a,p,n)\in T}[\max(0, ||f(a) - f(p)||_2^2 - ||f(a) - f(n)||_2^2 + margin)]$$

这里，a 表示锚点样本，p 表示正样本，n 表示负样本，T 表示所有可能的三元组集合，$margin$ 是一个预设的边界值，用于确保正样本对和负样本对之间有足够的距离。

（3）N-Pairs Loss
同时考虑多对正负样本

$$L = \sum_{i=1}^{N} \log(1 + \sum_{j=1, j\neq i}^{N} \exp(f(i)^T f(j) - f(i)^T f(i^+)))$$

这里，$f(i)$ 和 $f(i^+)$ 分别是同一类中的样本 i 及其匹配样本的特征表示，而 $f(j)$ 是其他类的样本特征表示。

（4）InfoNCE Loss （常见）
输入是一个锚点、一个正样本、多个负样本

InfoNCE Loss的公式

给定一个锚点样本 x，一个正样本 x^+（与锚点样本相似或在某种程度上相关的样本），以及 $N-1$ 个负样本 $\{x_1^-, x_2^-, ..., x_{N-1}^-\}$（与锚点样本不相似的样本），InfoNCE loss 的公式定义如下：

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp(sim(x, x^+)/\tau)}{\exp(sim(x, x^+)/\tau) + \sum_{i=1}^{N-1} \exp(sim(x, x_i^-)/\tau)}$$

其中，$sim(x, x')$ 是样本 x 和 x' 之间的相似度函数，通常使用点积或余弦相似度来计算。参数 τ 称为温度参数，用于控制相似度分数的缩放，影响损失函数的敏感度。

2.26

1.

自动属性赋值装饰器 Python

```python
from functools import wraps
from inspect import getfullargspec, isfunction
from itertools import starmap

def autoassign(*names, **kwargs):
    """
    autoassign(function) -> method
    autoassign(*argnames) -> decorator
    autoassign(exclude=argnames) -> decorator

    allow a method to assign (some of) its arguments as attributes of
    'self' automatically.  E.g.

    >>> class Foo(object):
    ...     @autoassign
    ...     def __init__(self, foo, bar): pass
    ...
    >>> breakfast = Foo('spam', 'eggs')
    >>> breakfast.foo, breakfast.bar
    ('spam', 'eggs')

    To restrict autoassignment to 'bar' and 'baz', write:

        @autoassign('bar', 'baz')
        def method(self, foo, bar, baz): ...

    To prevent 'foo' and 'baz' from being autoassigned, use:

        @autoassign(exclude=('foo', 'baz'))
        def method(self, foo, bar, baz): ...
    """
    if kwargs:
        exclude, f = set(kwargs['exclude']), None
        sieve = lambda l:filter(lambda nv: nv[0] not in exclude, l)
    elif len(names) == 1 and isfunction(names[0]):
        f = names[0]
        sieve = lambda l:l
    else:
        names, f = set(names), None
        sieve = lambda l: filter(lambda nv: nv[0] in names, l)
    def decorator(f):
        fargnames, _, _, fdefaults, _, _, _ = getfullargspec(f)
        # Remove self from fargnames and make sure fdefault is a tuple
        fargnames, fdefaults = fargnames[1:], fdefaults or ()

```

```
            defaults = list(sieve(zip(reversed(fargnames), reversed(fdefaults)
    )))
            @wraps(f)
            def decorated(self, *args, **kwargs):
                assigned = dict(sieve(zip(fargnames, args)))
                assigned.update(sieve(kwargs.items()))
                for _ in starmap(assigned.setdefault, defaults): pass
                self.__dict__.update(assigned)
                return f(self, *args, **kwargs)
            return decorated
        return f and decorator(f) or decorator
```

2.

优先经验回放 Python

```python
import numpy as np


class SumTree:
    """
    Story data with its priority in the tree.
    """
    data_pointer = 0# 用于指示下一条数据插入位置

    def __init__(self, capacity):
        self.capacity = capacity  # for all priority values
        self.tree = np.zeros(2 * capacity - 1)#用于存储优先值
        # [--------------Parent nodes-------------][-------leaves to recode priority-------]
        #             size: capacity - 1                       size: capacity
        self.data = list(np.zeros(capacity, dtype=object))  # 用于存储所有五元组(s,a,r,s_,done)
        # [--------------data frame-------------]
        #             size: capacity

    def add(self, p, transition):
        tree_idx = self.data_pointer + self.capacity - 1
        self.data[self.data_pointer] = transition  # update data_frame
        self.update(tree_idx, p)  # update tree_frame

        self.data_pointer += 1
        if self.data_pointer >= self.capacity:  # replace when exceed the capacity
            self.data_pointer = 0

    def update(self, tree_idx, p):
        change = p - self.tree[tree_idx]
        self.tree[tree_idx] = p
        # then propagate the change through tree
        while tree_idx != 0:    # 这种循环的写法比递归的写法快并且没有bug
            tree_idx = (tree_idx - 1) // 2
            self.tree[tree_idx] += change

    def get_leaf(self, v):
        """
        Tree structure and array storage:

        Tree index:
             0         -> storing priority sum
```

```
            / \
          1     2
         / \   / \
        3   4 5   6    -> storing priority for transitions

        Array type for storing:
        [0,1,2,3,4,5,6]

        注意，该树结构不一定是满二叉树，而是完全二叉树，因为叶子节点数量是capacity不一定是2的幂
        所有叶子节点存储样本以及优先值在data与tree数组中，非叶子节点存储优先值在tree数组中
        """
        parent_idx = 0
        while True:     # 这种循环的写法比递归的写法快
            cl_idx = 2 * parent_idx + 1         # this leaf's left and right kids
            cr_idx = cl_idx + 1
            if cl_idx >= len(self.tree):        # reach bottom, end search
                leaf_idx = parent_idx
                break
            else:       # downward search, always search for a higher priority node
                if v <= self.tree[cl_idx]:
                    parent_idx = cl_idx
                else:
                    v -= self.tree[cl_idx]
                    parent_idx = cr_idx

        data_idx = leaf_idx - self.capacity + 1
        return leaf_idx, self.tree[leaf_idx], self.data[data_idx]

    @property
    def total_p(self):
        return self.tree[0]  # the root


class Memory:
    '''
    负责与SumTree交互，实现五元组(s,a,r,s_,done)的存储、采样、更新工作
    '''

    epsilon = 0.01  # small amount to avoid zero priority
    alpha = 0.6  # [0~1] convert the importance of TD error to priority
    beta = 0.4  # importance-sampling, from initial value increasing to 1
    beta_increment_per_sampling = 0.001
    abs_err_upper = 1.  # clipped abs error
```

```
85
86
87      def __init__(self,
88                   memory_size,
89                   batch_size):
90          self.batch_size = batch_size
91          self.tree = SumTree(memory_size)
92          self.memory_num = 0#记录目前样本数，得知何时开始训练
93          self.memory_size = memory_size
94
95      def store(self, transition):
96          max_p = np.max(self.tree.tree[-self.tree.capacity:])
97          if max_p == 0:#回访池为空
98              max_p = self.abs_err_upper
99          self.tree.add(max_p, transition)   # set the max p for new transition p最大为1
100         if self.memory_num < self.memory_size:
101             self.memory_num += 1
102
103     def sample(self):
104         n = self.batch_size
105         b_idx, ISWeights = np.empty((n,), dtype=np.int32), np.empty((n, 1))#用于记录每个抽取到的样本的位置以及每个样本计算loss时的权重
106         b_memory = []
107         pri_seg = self.tree.total_p / n       # priority segment
108         self.beta = np.min([1., self.beta + self.beta_increment_per_sampling])  # max = 1
109
110         min_prob = np.min(self.tree.tree[-self.tree.capacity:]) / self.tree.total_p     # for later calculate ISweight
111         if min_prob == 0:
112             min_prob = 0.00001
113         for i in range(n):
114             a, b = pri_seg * i, pri_seg * (i + 1)
115             v = np.random.uniform(a, b)#在该区间中随机选一点
116             idx, p, data = self.tree.get_leaf(v)
117             prob = p / self.tree.total_p
118             ISWeights[i, 0] = np.power(prob/min_prob, -self.beta)
119             b_idx[i] = idx
120             b_memory.append(data)
121         return b_idx, b_memory, ISWeights
122
123     def batch_update(self, tree_idx, abs_errors):
124         abs_errors += self.epsilon  # convert to abs and avoid 0
125         clipped_errors = np.minimum(abs_errors, self.abs_err_upper)
126         ps = np.power(clipped_errors, self.alpha)#肯定都不会大于1
127         for ti, p in zip(tree_idx, ps):
              self.tree.update(ti, p)
```

3.

ndcg计算方式1 Python

```python
import numpy as np

def ndcg(r):
    """
    Compute NDCG (Normalized Discounted Cumulative Gain)

    Parameters:
    r (list): Relevance scores in rank order

    Returns:
    float: NDCG
    """
    r = np.asarray(r)
    dcg = np.sum(r / np.log2(np.arange(2, len(r) + 2)))
    idcg = np.sum(sorted(r, reverse=True) / np.log2(np.arange(2, len(r) + 2)))
    return dcg / idcg if idcg > 0 else 0

# Example usage
print("NDCG for [3, 2, 1]:", ndcg([3, 2, 1]))
```

ndcg计算方式2 Python

```python
#另一种工业界更常用的ndcg计算公式
import numpy as np

def ndcg(r):
    """
    Compute NDCG (Normalized Discounted Cumulative Gain) using 2^relevance - 1

    Parameters:
    r (list): Relevance scores in rank order

    Returns:
    float: NDCG
    """
    r = np.asarray(r)
    dcg = np.sum((np.power(2, r) - 1) / np.log2(np.arange(2, len(r) + 2)))
    idcg = np.sum((np.power(2, sorted(r, reverse=True)) - 1) / np.log2(np.arange(2, len(r) + 2)))
    return dcg / idcg if idcg > 0 else 0

# Example usage
print("NDCG for [3, 2, 1]:", ndcg([3, 2, 1]))
```

4.

折线变平滑 Python

```python
def moving_average(a, window_size):
    cumulative_sum = np.cumsum(np.insert(a, 0, 0))
    middle = (cumulative_sum[window_size:] - cumulative_sum[:-window_size]) / window_size
    r = np.arange(1, window_size - 1, 2)
    begin = np.cumsum(a[:window_size - 1])[::2] / r
    end = (np.cumsum(a[:-window_size:-1])[::2] / r)[::-1]
    return np.concatenate((begin, middle, end))
```

2.27

1.用bisect_left还是bisect_right就看等于target的值如何区分，如果与小于target的值处理相同则用bisect_right；如果与大于target的值处理相同则用bisect_left

2.28

1.数值/连续特征如何加到深度模型总结：

（1）归一化后直接加到深度模型中

归一化方式包括：直接归一化 $x' = \frac{x - min(x)}{max(x) - min(x)}$，或者通过bn层归一化 $\mathrm{BN}(x) = \gamma\left(\frac{x-\mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}}+\epsilon}}\right) + \beta$，

归一化后也可以再额外输入 $\sqrt{x}$ x^2 类的次线性与超线性的值(youtube dnn论文中用的)

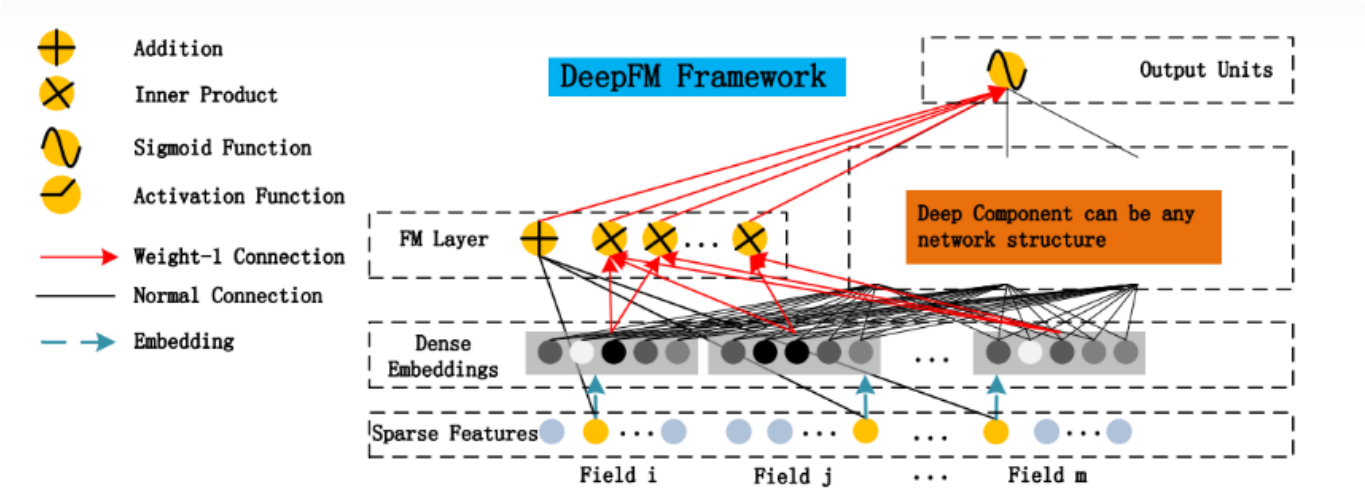
未归一化的数据会导致反向传播梯度的均值和方差过大，不利于网络的收敛

（2）离散化成为类别型特征，做embedding后输入网络，i.e.分桶

好处是：引入非线性，当label与该数值特征不是线性关系时，一个w * densefeature是不能很好刻画模型的，离散化后变成了对不同离散化值学习不同的系数，提升了非线性能力；过滤异常值，异常值会和其他值分到一个桶中，对模型影响变小

（3）给该数值特征本身一个embedding，而把数值作为权重乘这个特征的embedding作为该数值的embedding，只通过该数值本身区分生效（很少用）

2.DeepFM模型总结：



FM：离散特征做embedding，连续特征不用在FM（如果想用则需要分桶+embedding）

$$y(X) = \omega_0 + \sum^{n} \omega_i x_i + \sum^{n-1} \sum^{n} \omega_{ij} x_i x_j$$

FM模型去除二阶交互就是LR模型

一阶交互：给每个做one-hot后的神经元一个权重，如果是1则把权重加起来，所以图中是从sparse feature连出线的而不是从dense embeddings

二阶交互：dense embeddings两两点积(内积)再乘一个权重加起来

Deep：离散特征做embdding，连续特征做归一化拼到离散特征的embedding后面，两者共同输入到dnn