计算机视觉总结

SUMMARY OF COMPUTER VISION

(第1版)

LVSHUAILIN

VERSION 1

一. 数据结构与算法-LeetCode Hot 100

二. PYTHON: 1) NUMPY; 2) PANDAS; 3) PYTHON多进程; 4) PYTHON分布式; 5) PYTHON界面;

三. 深度学习: TensorFlow 2.0; PYTORCH;

四. 图像配准

五. 强化学习

六. OTHERS: 1) Model INFERENCE by EXE; 2) GIT; 3) DOCKER

LVSHUAILIN

2020年2月

目　　录

第1章　LeetCode Hot 100

Goals to Achieve

　　1. unordered_map.

§1.1 两数之和

> **HOT100 1.1　问题描述**
>
> 给定一个整数数组**nums** 和一个目标值**target**, 请你在该数组中找出和为目标值的那两个整数, 并返回他们的数组下标. 你可以假设每种输入只会对应一个答案. 但是, 你不能重复利用这个数组中同样的元素.
>
> 示例: 给定nums = [2, 7, 11, 15], target = 9; 因为nums[0] + nums[1] = 2 + 7 = 9;
>
> 所以返回[0, 1]
>
> `https://leetcode-cn.com/problems/two-sum`

1.1.1 知识点(unordered_map)

　　unordered_map内部是一个关联容器, 采用hash 表结构, 有快速检索的功能.

　　哈希表是通过key关键字直接访问对应value值的数据结构. 特点是键和值一一对应, 查找时间复杂度**O(1)**.

　　Example_1: unordered_map插入, 迭代遍历.

　　unordered_map example_1 code

```cpp
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;
int main()
{
```

```
7        unordered_map<string, double> umap;
8        umap["PI"] = 3.14;
9        umap.insert(make_pair("a", 2.1));
10
11       // find in umap
12       string key = "PI";
13       if (umap.find(key) == umap.end())
14           cout << "cannot find PI" << endl;
15       else
16           cout << "find " << umap.find(key)->first << " = " << umap.find(key)->second << endl;
17
18       // iterator of umap
19       cout << "entire unorded_map is:"<<endl;
20       unordered_map<string, double>::iterator itr;
21       for (itr = umap.begin(); itr != umap.end(); ++itr)
22           cout << " (" << itr->first << "," << itr->second << ") " << endl;
23       system("pause");
24       return 0;
25   }
```

---

output:
find PI = 3
all elements are:
(PI,3.14)
(a,2.1)

Example_2: 利用unordered_map输出一段文字中重复单词的个数

unordered_map example_2 code

```
1    #include <iostream>
2    #include <unordered_map>
3    #include <string>
4    #include <sstream>
5
6    using namespace std;
7
8    void printWordFreq(const string& str)
9    {
10       unordered_map<string, int> wordFreq;
11       string word;
12       stringstream ss(str);
```

```
13      while (ss >> word)
14          wordFreq[word]++;
15
16      cout << "all elements are:" << endl;
17      for (auto u : wordFreq)
18          cout << "(" << u.first << "," << u.second << ")" << endl;
19  }
20
21  int main()
22  {
23      string str = "studies very very hard";
24      printWordFreq(str);
25      return 0;
26  }
```

---

output:
all elements are:
(studies, 1)
(very, 2)
(hard, 1)

1.1.2 解题代码

```
1   #include <iostream>
2   #include <unordered_map>
3   #include <vector>
4   using namespace std;
5
6   vector<int> twoSum(vector<int>& nums, int target)
7   {
8       unordered_map<int, int> map;
9       vector<int> result={};
10      int n = (int)nums.size();
11      for(int i = 0; i < n; ++i) {
12          auto p = map.find(target-nums[i]);
13          if(p != map.end()) {
14          result.push_back(p->second);
15          result.push_back(i);
16          }
```

```
17        map[nums[i]] = i;
18      }
19      return result;
20  }
21
22  int main()
23  {
24      vector< int > nums = {2,7,11,15};
25      vector<int> result;
26      result = twoSum(nums,9);
27      cout<<"["<<result[0] << "," <<result[1]<<"]"<<endl;
28      return 0;
29  }
```

§1.2 两数相加

> **HOT100 1.2 问题描述**
>
> 给出两个非空的链表用来表示两个非负的整数. 其中, 它们各自的位数是按照逆序的方式存储的, 并且它们的每个节点只能存储一位数字. 如果, 我们将这两个数相加起来, 则会返回一个新的链表来表示它们的和.
>
> 您可以假设除了数字0之外, 这两个数都不会以0开头.
>
> 示例: 输入$(2->4->3)+(5->6->4)$, 输出: $7->0->8$, 原因: $342+465=807$
>
> https://leetcode-cn.com/problems/add-two-numbers

1.2.1 知识点(linked list)

这里用c++ 链表来解决

Example_1: 创建链表并初始化

linked list example_1 code

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Node{
6  public:
```

```
7        int data;
8        Node* next;
9    };
10
11   int main()
12   {
13       Node* head = nullptr;
14       Node* second = nullptr;
15       Node* third = nullptr;
16
17       head = new Node();
18       head->data = 1;
19
20       second = new Node();
21       second->data = 2;
22
23       third = new Node();
24       third->data = 3;
25
26       cout << head->data << " " << second->data << " " << third->data << endl;
27
28       delete head;
29       delete second;
30       delete third;
31       return 0;
32   }
```

```
output:
1 2 3
```

Example_2: 打印链表中的所有元素

linked list example_2 code

```
1    #include <iostream>
2
3    using namespace std;
4
5    class Node{
6    public:
7        int data;
8        Node* next;
```

```cpp
9  };
10
11 void PrintLinkedList(Node* head)
12 {
13     Node* temp = head;
14     while (temp != nullptr) {
15         cout << temp->data << " ";
16         temp = temp->next;
17     }
18     cout << endl;
19 }
20
21 int main()
22 {
23     Node* head = nullptr;
24     Node* second = nullptr;
25     Node* third = nullptr;
26
27     head = new Node();
28     second = new Node();
29     third = new Node();
30
31     head->data = 1;
32     head->next = second;
33
34     second->data = 2;
35     second->next = third;
36
37     third->data = 3;
38     third->next = nullptr;
39
40
41     PrintLinkedList(head);
42
43     delete head;
44     delete second;
45     delete third;
46     return 0;
47 }
```

output:
1 2 3

Example_3: 链表插入节点

linked list example_3 code

```cpp
#include <iostream>

using namespace std;

class Node{
public:
    int data;
    Node* next;
};

// 在链表前面插入节点
void push(Node** head_ref, int newData)
{
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = (*head_ref);
    (*head_ref) = newNode;
}

// 在节点后面插入节点
void insertAfter(Node** prev_node, int newData)
{
    if ((*prev_node) == nullptr) {
        cout << "the previous node cannot be nullptr" << endl;
        return;
    }

    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = (*prev_node)->next;
    (*prev_node)->next = newNode;
}

// 在尾节点后插入节点
void append(Node** head_ref, int newData)
```

```
36  {
37      Node* newNode = new Node();
38      newNode->data = newData;
39      newNode->next = nullptr;
40      if ((*head_ref) == nullptr) {
41          (*head_ref) = newNode;
42          return;
43      }
44
45      Node* move = (*head_ref);
46      while (move->next != nullptr) {
47          move = move->next;
48      }
49      move->next = newNode;
50  }
51
52  // 打印链表
53  void PrintLinkedList(Node* head)
54  {
55      Node* temp = head;
56      while (temp != nullptr) {
57          cout << temp->data << "  ";
58          temp = temp->next;
59      }
60      cout << endl;
61  }
62
63  void destroyLinkedList(Node** head_ref) {
64      Node* move = (*head_ref);
65      Node* next = nullptr;
66      while (move != nullptr) {
67          next = move->next;
68          delete move;
69          move = next;
70      }
71      (*head_ref) = nullptr;
72  }
73
74  int main()
75  {
```

```
76      Node* head = nullptr;
77
78      append(&head, 6);
79
80      push(&head, 7);
81
82      push(&head, 1);
83
84      append(&head, 4);
85
86      insertAfter(&(head->next), 8);
87
88      cout << "linked list is: ";
89      PrintLinkedList(head);
90      destroyLinkedList(&head);
91      return 0;
92  }
```

output:
linked list is: 1 7 8 6 4

1.2.2 解题代码

```
1   #include <iostream>
2   using namespace std;
3
4   struct ListNode {
5       int val;
6       ListNode *next;
7       ListNode(int x) : val(x), next(NULL) {}
8   };
9
10  ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
11      int len1 = 1;//记录的长度l1
12      int len2 = 1;//记录的长度l2
13      ListNode* p = l1;
14      ListNode* q = l2;
15      while (p->next != NULL)//获取的长度l1
16      {
```

```
17          len1++;
18          p = p−>next;
19      }
20      while (q−>next != NULL)//获取的长度l2
21      {
22          len2++;
23          q = q−>next;
24      }
25      if (len1 > len2)//较长，在末尾补零l1l2
26      {
27          for (int i = 1; i <= len1 − len2; i++)
28          {
29              q−>next = new ListNode(0);
30              q = q−>next;
31          }
32      }
33      else//较长，在末尾补零l2l1
34      {
35          for (int i = 1; i <= len2 − len1; i++)
36          {
37              p−>next = new ListNode(0);
38              p = p−>next;
39          }
40      }
41      p = l1;
42      q = l2;
43      bool count = false;//记录进位
44      ListNode∗ l3 = new ListNode(−1);//存放结果的链表
45      ListNode∗ w = l3;//的移动指针l3
46      int i = 0;//记录相加结果
47      while (p != NULL && q != NULL)
48      {
49          i = count + p−>val + q−>val;
50          w−>next = new ListNode(i % 10);
51          count = i >= 10 ? true : false;
52          w = w−>next;
53          p = p−>next;
54          q = q−>next;
55      }
56      if (count)//若最后还有进位
```

```
57          {
58              w->next = new ListNode(1);
59              w = w->next;
60          }
61          return l3->next;
62  }
63
64  void printLinkedList(ListNode* head)
65  {
66      ListNode* move = head;
67      while (move != nullptr) {
68          cout << move->val << "  ";
69          move = move->next;
70      }
71  }
72
73  int main()
74  {
75  #if 1
76      ListNode* l1 = new ListNode(2);
77      ListNode* l1_1 = new ListNode(4);
78      ListNode* l1_2 = new ListNode(3);
79
80      l1->next = l1_1;
81      l1_1->next = l1_2;
82
83
84      ListNode* l2 = new ListNode(5);
85      ListNode* l2_1 = new ListNode(6);
86      ListNode* l2_2 = new ListNode(4);
87      l2->next = l2_1;
88      l2_1->next = l2_2;
89  #endif
90
91  #if 0
92      ListNode* l1 = new ListNode(5);
93      ListNode* l2 = new ListNode(5);
94  #endif
95
96      ListNode* result = addTwoNumbers(l1, l2);
```

```
97        printLinkedList(result);
98        return 0;
99   }
```

output:
7 0 8

§1.3 无重复字符的最长子串

> ### HOT100 1.3   问题描述
>
> 给定一个字符串, 请你找出其中不含有重复字符的最长子串的长度.
>
> **示例1**:
>
> 输入: "abcabcb"
>
> 输出: 3
>
> 解释: 因为无重复字符的最长子串是"abc", 所以其长度为3.
>
> **示例2**:
>
> 输入: "bbbbb"
>
> 输出: 1
>
> 解释: 因为无重复字符的最长子串是"b", 所以其长度为1.
>
> **示例3**:
>
> 输入: "pwwkew"
>
> 输出: 3
>
> 解释: 因为无重复字符的最长子串是"wke", 所以其长度为3. 请注意, 你的答案必须是子串的长度, "pwke"是一个子序列, 不是子串.
>
> https://leetcode-cn.com/problems/longest-substring-without-repeating-characters

1.3.1 知识点(double pointer algorithm和unordered_set)

c++提供两种关联型数据结构, 1) 树型结构, 如: map, set; 2) hash结构, 如: unordered_map, unordered_set. map和set是有序的, 其他两个是无序的.

1.3.2 解题思路

https://cloud.tencent.com/developer/article/1377650

这道题主要用到思路是: 滑动窗口

什么是滑动窗口?

其实就是一个队列, 比如例题中的abcabcbb, 进入这个队列(窗口)为abc满足题目要求, 当再进入a, 队列变成了abca, 这时候不满足要求. 所以, 我们要移动这个队列!

如何移动?

我们只要把队列的左边的元素移出就行了, 直到满足题目要求!

一直维持这样的队列, 找出队列出现最长的长度时候, 求出解!

时间复杂度: **O(n)**

1.3.3 解题代码

```cpp
#include <iostream>
#include <string>
#include <unordered_set>
#include <algorithm> // max, min

using namespace std;

int lengthOfLongestSubstring(string s) {
    if (s.size() == 0) return 0;
    unordered_set<char> lookup;
    int maxStr = 0;
    int left = 0;
    for (int i = 0; i < s.size(); i++) {
        while (lookup.find(s[i]) != lookup.end()) {
            lookup.erase(s[left]);
            left++;
        }
        maxStr = max(maxStr, i - left + 1);
        lookup.insert(s[i]);
    }
    return maxStr;
}

int main()
```

```
25  {
26      string str = "abcabcb";
27      cout << lengthOfLongestSubstring(str) << endl;
28      return 0;
29  }
```

output:
3

§1.4 寻找两个有序数组的中位数

> **HOT100 1.4　问题描述**
>
> 给定两个大小为m和n的有序数组nums1和nums2.
>
> 请你找出这两个有序数组的中位数, 并且要求算法的时间复杂度为**O(log(m + n))**.
>
> 你可以假设nums1和nums2不会同时为空.
>
> **示例1**:
>
> nums1 = [1, 3]
>
> nums2 = [2]
>
> 则中位数是2.0
>
> **示例2**:
>
> nums1 = [1, 2]
>
> nums2 = [3, 4]
>
> 则中位数是(2 + 3)/2 = 2.5
>
> https://leetcode-cn.com/problems/median-of-two-sorted-arrays

1.4.1 知识点(二分查找算法)

用二分查找算法, 也叫做折半查找算法.

Example_1: 二分查找

二分查找算法example_1 code

```
1  // 二分查找– 折半查找
2  int search(int arr[], int key, int left, int right)
3  {
4      while (left <= right)
```

```cpp
    {
        int mid = left + (right − left) / 2;
        if (key < arr[mid])
            right = mid − 1;
        else if (key > arr[mid])
            left = mid + 1;
        else
            return mid;
    }
    return −1;
}

int main()
{
    int arr[] = { 0,2 ,3,4};
    int value = 3;

    // left Index of the array
    int left = 0;

    // right Index of the array
    int right = sizeof(arr) / sizeof(arr[0]) − 1;

    cout << "left: " << left << ", right: " << right << endl;

    int ret = search(arr, value, left, right);
    if (ret == −1)
        printf("cannot find the value");
    else
        printf("found the value, the index is: %d\n", ret);
    system("pause");
    return 0;
}
```

output:
left: 0, right: 3
found the value, the index is: 2

1.4.2 解题思路

这道题让我们求两个有序数组的中位数，而且限制了时间复杂度为O(log (m+n))，看到这个时间复杂度，自然而然的想到了应该使用二分查找法来求解。那么回顾一下中位数的定义，如果某个有序数组长度是奇数，那么其中位数就是最中间那个，如果是偶数，那么就是最中间两个数字的平均值。这里对于两个有序数组也是一样的，假设两个有序数组的长度分别为m和n，由于两个数组长度之和m+n 的奇偶不确定，因此需要分情况来讨论，对于奇数的情况，直接找到最中间的数即可，偶数的话需要求最中间两个数的平均值。为了简化代码，不分情况讨论，我们使用一个小trick，我们分别找第(m+n+1) / 2 个，和(m+n+2) / 2 个，然后求其平均值即可，这对奇偶数均适用。假如m+n 为奇数的话，那么其实(m+n+1) / 2 和(m+n+2) / 2 的值相等，相当于两个相同的数字相加再除以2，还是其本身。

这里我们需要定义一个函数来在两个有序数组中找到第K个元素，下面重点来看如何实现找到第K个元素。首先，为了避免产生新的数组从而增加时间复杂度，我们使用两个变量i和j分别来标记数组nums1和nums2的起始位置。然后来处理一些边界问题，比如当某一个数组的起始位置大于等于其数组长度时，说明其所有数字均已经被淘汰了，相当于一个空数组了，那么实际上就变成了在另一个数组中找数字，直接就可以找出来了。还有就是如果K=1的话，那么我们只要比较nums1和nums2的起始位置i和j上的数字就可以了。难点就在于一般的情况怎么处理？因为我们需要在两个有序数组中找到第K个元素，为了加快搜索的速度，我们要使用二分法，对K二分，意思是我们需要分别在nums1和nums2中查找第K/2个元素，注意这里由于两个数组的长度不定，所以有可能某个数组没有第K/2个数字，所以我们需要先检查一下，数组中到底存不存在第K/2个数字，如果存在就取出来，否则就赋值上一个整型最大值。如果某个数组没有第K/2个数字，那么我们就淘汰另一个数字的前K/2个数字即可。有没有可能两个数组都不存在第K/2个数字呢，这道题里是不可能的，因为我们的K不是任意给的，而是给的m+n的中间值，所以必定至少会有一个数组是存在第K/2个数字的。最后就是二分法的核心啦，比较这两个数组的第K/2小的数字midVal1和midVal2的大小，如果第一个数组的第K/2个数字小的话，那么说明我们要找的数字肯定不在nums1中的前K/2个数字，所以我们可以将其淘汰，将nums1的起始位置向后移动K/2个，并且此时的K也自减去K/2，调用递归。反之，我们淘汰nums2中的前K/2个数字，并将nums2的起始位置向后移动K/2个，并且此时的K也自减去K/2，调用递归即可。

1.4.3 解题代码

```
/*分清 起始位置和第几个元素*/
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int findKthNumber(vector<int>& nums1, int i, vector<int>& nums2, int j, int k) {
    if (i >= nums1.size()) return nums2[j + k - 1];
    if (j >= nums2.size()) return nums1[i + k - 1];
    //if(k ==1) return (double(nums1[i] + nums2[j]));wrong
```

```cpp
11        if (k == 1) return min(nums1[i], nums2[j]);
12        //查找有没有k个元素的位置/2 i + k/2 −1
13        int midVal1 = (i + k / 2 − 1 < nums1.size()) ? nums1[i + k / 2 − 1] : INT_MAX;
14        int midVal2 = (j + k / 2 − 1 < nums2.size()) ? nums2[j + k / 2 − 1] : INT_MAX;
15        if (midVal1 < midVal2)
16            return findKthNumber(nums1, i + k / 2, nums2, j, k − k / 2);
17        else
18            return findKthNumber(nums1, i, nums2, j + k / 2, k − k / 2);
19    }
20    class Solution {
21    public:
22        double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
23            int m = nums1.size(), n = nums2.size();
24            int left = (m + n + 1) / 2, right = (m + n + 2) / 2;
25            return (findKthNumber(nums1, 0, nums2, 0, left) + findKthNumber(nums1, 0, nums2, 0, right)) / 2.0;
26        }
27    };
```

output:
Null

第2章 Bayesian-Python

<div style="background:#e8f4f8;padding:10px;">

学习目标与要求

1. .
2. .
3. .
4. .

</div>

§ 2.1 Probalilities

A common and useful conceptualization in statistics is to think that data was generated from some probability distribution with unobserved parameters.

2.1.1 Gaussian Distribution

$$pdf(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns

mu_params = [−1, 0, 1]
sd_params = [0.5, 1, 1.5]
x = np.linspace(−7, 7, 100)
f, ax = plt.subplots(len(mu_params), len(sd_params), sharex=True, sharey=True)
for i in range(3):
    for j in range(3):
```

```
12          mu = mu_params[i]
13          sd = sd_params[j]
14          y = stats.norm(mu, sd).pdf(x)
15          ax[i, j].plot(x,y)
16          ax[i, j].plot(0, 0, label="$\\mu$ = {:3.2f}\n$\\sigma$={:3.2f}".format (mu, sd), alpha=0)
17          ax[i, j].legend(fontsize=12)
18      ax[2,1].set_xlabel('$x$', fontsize=16)
19      ax[1,0].set_ylabel('$pdf(x)$', fontsize=16)
20      plt.tight_layout()
```
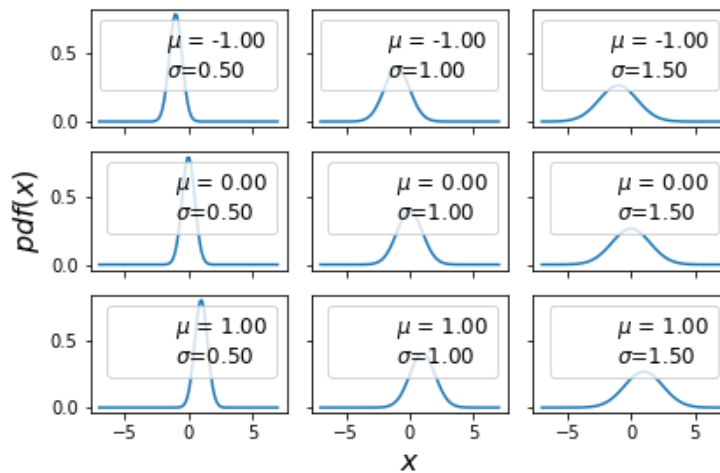

图 2.1   Gaussion Distribution

2.1.2 Coin-Flipping Problen

We will answer this question in a Bayesian setting. We will need data and a probabilistic model.

Data: we will assume that we have already tossed a coin a number of times and we have recorded the number of observed head, so the data part is done.

Model will be discussed soon.

2.1.2.1 The general model

The firs thing we will do is generalize the concept of bias. We will say that a coin with a bias of 1 will always land heads, one with a bias of 0 will always land tails, and one with a bias of 0.5 will land half of the time heads and half of the time tails. To represent the bias, we will use the parameter θ, and to represent the total number of heads for an N number of tosses, we will use the variable y. According to Bayes' theorem we have the following formula:

$$p(\theta|y) \propto p(y|\theta)p(\theta)$$

Notice that we need to specify which prior $p(\theta)$ and likelihood $p(y|\theta)$ we will use. Let's start with the likelihood.

2.1.2.2 Choosing the likelihood

Let's assume that a coin toss does not affect other tosses, that is, we are assuming coin tosses are independent of each other. Let's also assume that only two outcomes are possible, heads or tails. Given these assumptions, a good candidate for the likelihood is the binomial distribution (二项分布):

$$p(y|\theta) = \frac{N!}{N!(N-y)!}\theta^y(1-\theta)^{N-y}$$

This is a discrete distribution returning the probability of getting y heads (or in general, success) out of N coin tosses (or in general, trials or experiments) given a fixed value of θ. The following code generates 9 binomial distributions; each subplot has its own legend indicating the corresponding parameters:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from scipy import stats
4  import seaborn as sns
5
6  n_params = [1,2,4]
7  p_params = [0.25, 0.5, 0.75]
8  x = np.arange(0, max(n_params) + 1)
9  f, ax = plt.subplots(len(n_params), len(p_params), sharex=True, sharey=True)
10 for i in range(3):
11     for j in range(3):
12         n = n_params[i]
13         p = p_params[j]
14         y = stats.binom(n=n, p=p).pmf(x)
15         ax[i,j].vlines(x, 0, y, colors='b', lw=5)
16         ax[i,j].set_ylim(0,1)
17         ax[i,j].plot(0,0,label="n = {:3.2f}\np = {:3.2f}".format(n,p), alpha=0)
18         ax[i,j].legend(fontsize=12)
19 ax[2,1].set_xlabel('$\\theta$', fontsize=14)
20 ax[1,0].set_ylabel('$p(y|\\theta)$', fontsize=14)
21 ax[0,0].set_xticks(x)
22 plt.savefig('binomial_distribution.png')
```

The binomial distribution is also a reasonable choice for the likelihood. Intuitively, we can see that θ indicates how likely it is that we will obtain a head when tossing a coin, and we have observed that event y times. Following the same line of reasoning we get that $1-\theta$ is the chance of getting a tail, and that event has occurred $N-y$ times.

OK, so if we know θ, the binomial distribution will tell us the expected distribution of head. The only problem is that we do not kno θ!. But do not despair; in Bayesian statistics, every time we do not know the value of a parameter, we put a prior on it, so let's move on and choose a prior.
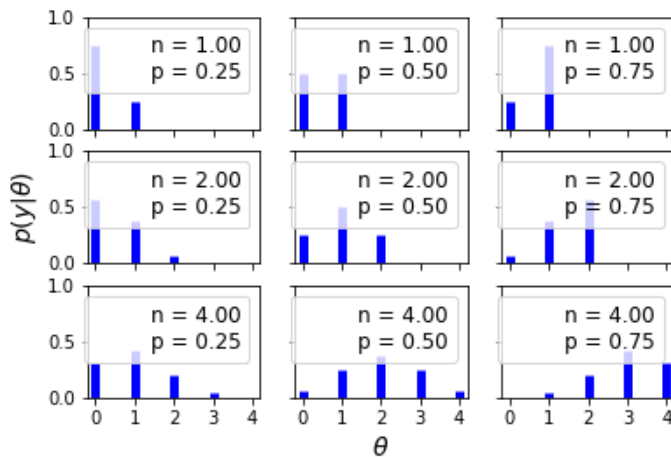
图 2.2　binomial_distribution

2.1.2.3 Choosing the prior

As a prior we will use a beta distribution (贝塔分布), which is a very common distribution in Bayesian statistics and looks like this:

$$p(\theta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

The first term is a normalization constant that ensures the distribution integrates to 1.

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns

params = [0.5, 1, 2, 3]
x = np.linspace(0, 1, 100)
f, ax = plt.subplots(len(params), len(params), sharex=True, sharey=True)
for i in range(4):
    for j in range(4):
        a = params[i]
        b = params[j]
        y = stats.beta(a, b).pdf(x)
        ax[i,j].plot(x, y)
        ax[i,j].plot(0,0,label="$\\alpha$ = {:3.2f}\n$\\beta$={:3.2f}".format(a,b), alpha=0)
        ax[i,j].legend(fontsize=12)
ax[3,0].set_xlabel('$\\theta$', fontsize=14)
ax[0,0].set_ylabel('$p(\\theta)$', fontsize=14)
```

```
19  plt.savefig('beta_distribution.png')
```
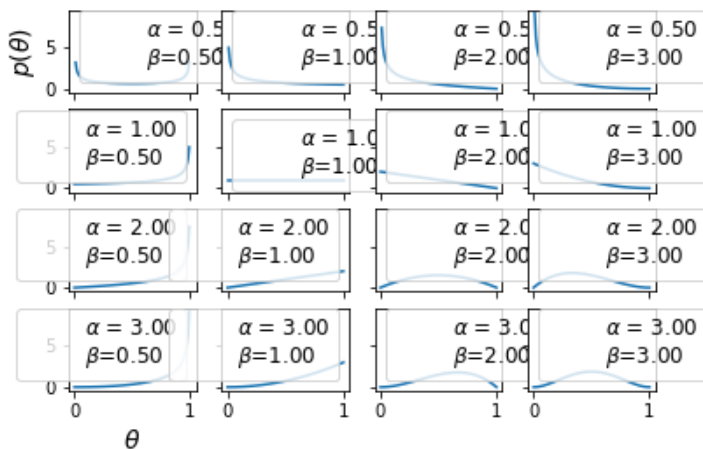

图 2.3 beta_distribution

Why are we using the beta distribution for our model? 1) One reason is that the beta distribution is restricted to be between 0 and 1, in the same way our parameter θ is. 2) Another reason is its versatility (通用性). As we can see in the preceding figure, the distribution adopts several shapes, including a uniform distribution, Gaussian-like distributions, U-like distributions, and so on. 3) A third reason is that the beta distribution is the conjugate prior (共轭先验) of the binomial distribution (which we are using as the likelihood). A conjugate prior of a likelihood is a prior that, when used in combination with the given likelihood, returns a posterior with the same functional form as the prior. There are other pairs of conjugate priors, for example, the Gaussion distribution is the conjugate prior of itself.

For many years, Bayesian analysis was restricted to the use of conjugate priors. Conjugacy ensures mathematical tractability of the posterior, which is important given that common problem in Bayesian statics is to end up with a posterior we cannot solve analytically. This was a deal beaker before the development of suitable computational methods to solve any possible posterior.

However, modern computational methods to solve Bayesian problems whether we choose conjugate priors or not.

2.1.2.4 Getting the posterior

The Bayes' theorem says that the posterior is proportional to the likelihood times the prior:

$$p(\theta|y) \propto p(y|\theta)p(\theta)$$

which turns out to be

$$p(\theta|y) \propto \frac{N!}{N!(N-y)!}\theta^y(1-\theta)^{N-y}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

To our practical concerns we can drop all the terms that do not depend on θ and our results will still be valid. So we can write the following:

$$p(\theta|y) \propto \theta^y(1-\theta)^{N-y}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Reordering it, we have

$$p(\theta|y) \propto \theta^{\alpha-1+y}(1-\theta)^{\beta-1+N-y}$$

We will see that this expression has the same functional form of a beta distribution (except for the normalization) with $\alpha_{posterior} = \alpha_{perior} + y$ and $\beta_{posterior} = \beta_{prior} + N - y$, which means that the posterior for our problem is the beta distribution:

$$p(\theta|y) = Beta(\alpha_{prior} + y, \beta_{prior} + N - y)$$

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns


theta_real = 0.35
trials = [0, 1, 2, 3, 4, 8, 16, 32, 50, 150]
data = [0, 1, 1, 1, 1, 4, 6, 9, 13, 48]


beta_params = [(1,1), (0.5,0.5), (20,20)]
dist = stats.beta
x = np.linspace(0, 1, 100)


for idx, N in enumerate(trials):
    if idx == 0:
        plt.subplot(4, 3, 2)
    else:
        plt.subplot(4, 3, idx+3)
    y = data[idx]
    for (a_prior, b_prior), c in zip(beta_params, ('b', 'r', 'g')):
        p_theta_given_y = dist.pdf(x, a_prior+y, b_prior+N-y)
        plt.fill_between(x, 0, p_theta_given_y, color=c, alpha=0.6)

    plt.axvline(theta_real, ymax=0.3, color='k')
    plt.plot(0,0,label="{:d} experiments\n{:d} heads".format(N,y),alpha=0)
    plt.xlim(0,1)
    plt.ylim(0,12)
    plt.xlabel(r'$\theta$')
    plt.legend()
    plt.gca().axes.get_yaxis().set_visible(False)
plt.tight_layout()
plt.savefig("posterior.png")
```
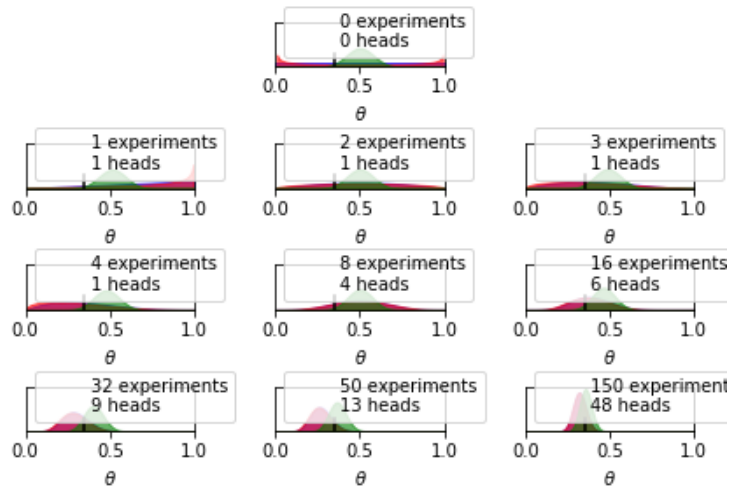
图 2.4   posterior

2.1.2.5 Model notation and visualization

A common notation to succinctly represent probabilistic is as follows:

$$\checkmark\, \theta \sim Beta(\alpha, \beta)$$

$$\checkmark\, y \sim Bin(n = 1, p = \theta)$$

第3章   深度学习

Goals to Achieve

1. pytorch basics

2. pytorch projects

§ 3.1 Pytorch

打印模型结构

pip install torchsummary

summary(model, (3, 32, 32))

print(model)