**1-**

sample\_size = header[0] + (header[1] << 8) + (header[2] << 16) + (header[3] << 24)

tagcode = header[5] + (header[4] << 8)

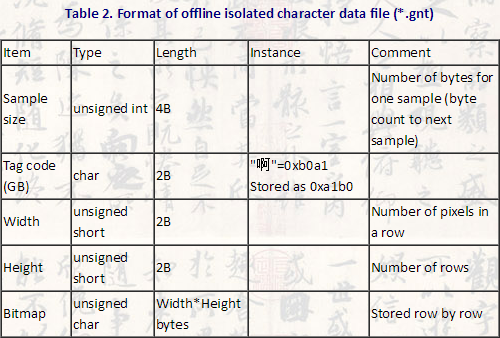
width = header[6] + (header[7] << 8)

height = header[8] + (header[9] << 8)

**if** header\_size + width \* height != sample\_size:

**break**

**gnt文件格式分析：**



如图所示，文件的前4个字节记录了第一个字符数据与下一个字符数据之间的字节数，比如这4个字节是35656的话，第二个字符的数据就是从第35657个字节开始（下标35656）。接下来的两个字节存储该字符的GBK编码，再下面的4个字节记录了字符图片的宽度和高度，然后就是字符图片的数据（按行存储）。

storage: header[5] [4]

a1 b0

tagcode = header[5] + (header[4] << 8) ##b0a1

**2- yield image, tagcode**

实现﻿yield的功能类似于return,但是不同之处在于它返回的是生成器。

如果一个函数包含yield关键字，这个函数就会变为一个生成器。

生成器并不会一次返回所有结果，而是每次遇到yield关键字后返回相应结果，并保留函数当前的运行状态，等待下一次的调用。

在一个 generator function 中，如果没有 return，则默认执行至函数完毕，如果在执行过程中 return，则直接抛出 StopIteration 终止迭代。

**3- tagcode\_unicode = struct.pack('>H', tagcode).decode('gb2312')**

pack(fmt, v1, v2, ...)#  按照给定的格式(fmt)，把数据封装成字符串(实际上是类似于c结构体的字节流)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FORMAT | C TYPE | PYTHON TYPE | STANDARD SIZE | NOTES |
| x | pad byte | no value |  |  |
| c | char | string of length 1 | 1 |  |
| b | signed char | integer | 1 | (3) |
| B | unsigned char | integer | 1 | (3) |
| ? | \_Bool | bool | 1 | (1) |
| h | short | integer | 2 | (3) |
| H | unsigned short | integer | 2 | (3) |
| i | int | integer | 4 | (3) |
| I | unsigned int | integer | 4 | (3) |
| l | long | integer | 4 | (3) |
| L | unsigned long | integer | 4 | (3) |
| q | long long | integer | 8 | (2), (3) |
| Q | unsigned long long | integer | 8 | (2), (3) |
| f | float | float | 4 | (4) |
| d | double | float | 8 | (4) |
| s | char[] | string |  |  |
| p | char[] | string |  |  |
| P | void \* | integer |  | (5), (3) |

为了同c中的结构体交换数据，还要考虑有的c或c++编译器使用了字节对齐，通常是以4个字节为单位的32位系统，故而struct根据本地机器字节顺序转换.可以用格式中的第一个字符来改变对齐方式.定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| CHARACTER | BYTE ORDER | SIZE | ALIGNMENT |
| @ | native | native | native |
| = | native | standard | none |
| < | little-endian | standard | none |
| > | big-endian | standard | none |
| ! | network (= big-endian) | standard | none |

=>b=struct.pack('>H', 0xb0a1)

=>b = '\xb0\xa1' ## H : unsigned short, 用2bytes表示

**4 - One Hot编码/独热编码**

整数编码：假设我们有一个带有'red'和'green'值的标签序列。我们可以将'red'的整数值分配为0，'green'的整数值为1。只要我们总是将这些数字分配给这些标签，这称为整数编码。

接下来，我们可以创建一个二进制向量来表示每个整数值。对于2个可能的整数值，向量的长度为2。

编码为0的“红色”标签将用二进制向量[1,0]表示，其中第0个索引被标记为值1。然后，编码为1的“绿色”标签将用一个二进制向量[0，1]，其中第一个索引被标记为1。

我们想让网络更具表现力，为每个可能的标签值提供概率式数字。这有助于进行问题网络建模。当输出变量使用one-hot编码时，它可以提供比单个标签更准确的一组预测。

**5- batch\_size, iteration, epoch**

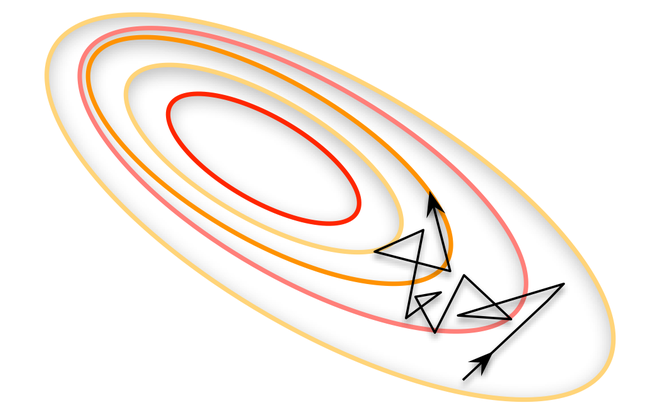
- epoch: 当一个完整的数据集通过了神经网络一次并且返回了一次，这个过程称为一次epoch。

- iteration: 1个iteration等于使用batchsize个样本训练一次, 即batch\_size个样本正向通过神经网络后，综合计算其损失函数，再反向权值更新, 总共算作一个iteration。在一个epoch中，number of iteration = 样本总数/batchsize.（或者理解iteration为w和b的一次更新）

- batch\_size: 以batch\_size的大小将数据输入深度学习的网络中，然后计算这个batch的所有样本的平均损失，即代价函数是所有样本的平均,再反向权值更新.

batchsize影响：如图，线性神经元在均方误差代价函数的错误面是一个抛物面，横截面是椭圆。对于多层神经元、非线性网络，在局部依然近似是抛物面。

如果batchsize过小（例如batchsize=1), 每次代价函数的修正方向以各自样本的梯度方向修正，横冲直撞各自为政，难以达到收敛。



合理增大batchsize的作用：跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快；在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。

机器学习中参数更新的方法有三种：

（1）Batch Gradient Descent，批梯度下降，遍历全部数据集计算一次损失函数，进行一次参数更新，这样得到的方向能够更加准确的指向极值的方向，但是计算开销大，速度慢；

（2）Stochastic Gradient Descent，随机梯度下降，对每一个样本计算一次损失函数，进行一次参数更新，优点是速度快，缺点是方向波动大，忽东忽西，不能准确的指向极值的方向，有时甚至两次更新相互抵消；

（3）Mini-batch Gradient Decent，小批梯度下降，前面两种方法的折中，把样本数据分为若干批，分批来计算损失函数和更新参数，这样方向比较稳定，计算开销也相对较小。Batch Size就是每一批的样本数量。