1. 将triplet loss和quantization loss加在一起训练，但是效果不好。其中的量化损失的最大值在5000左右，我给每个量化损失除了1000，也可以算是归一化。但是正规的归一化应该是除以最大值

While epoch < epochs:

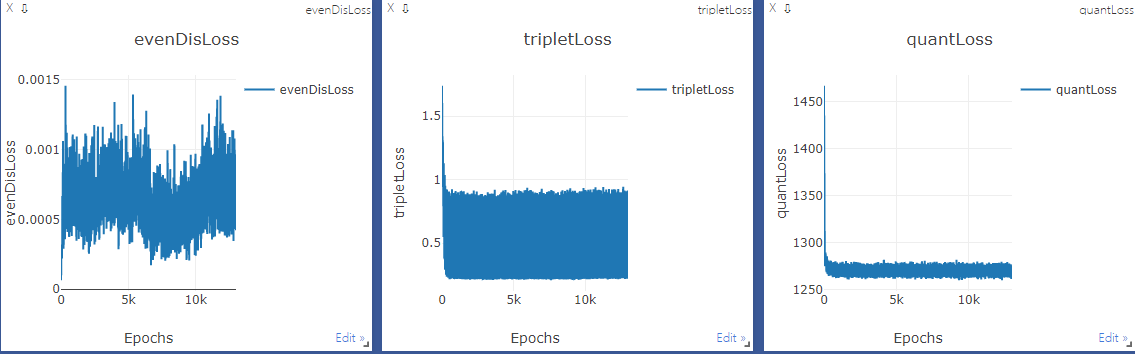
while batch < batches:

Fix W update ba;

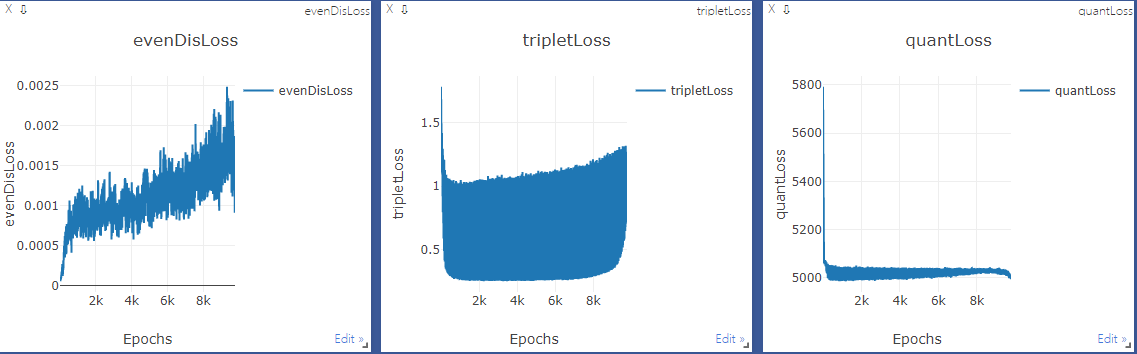
while iter < max\_iter:

Fix ba;

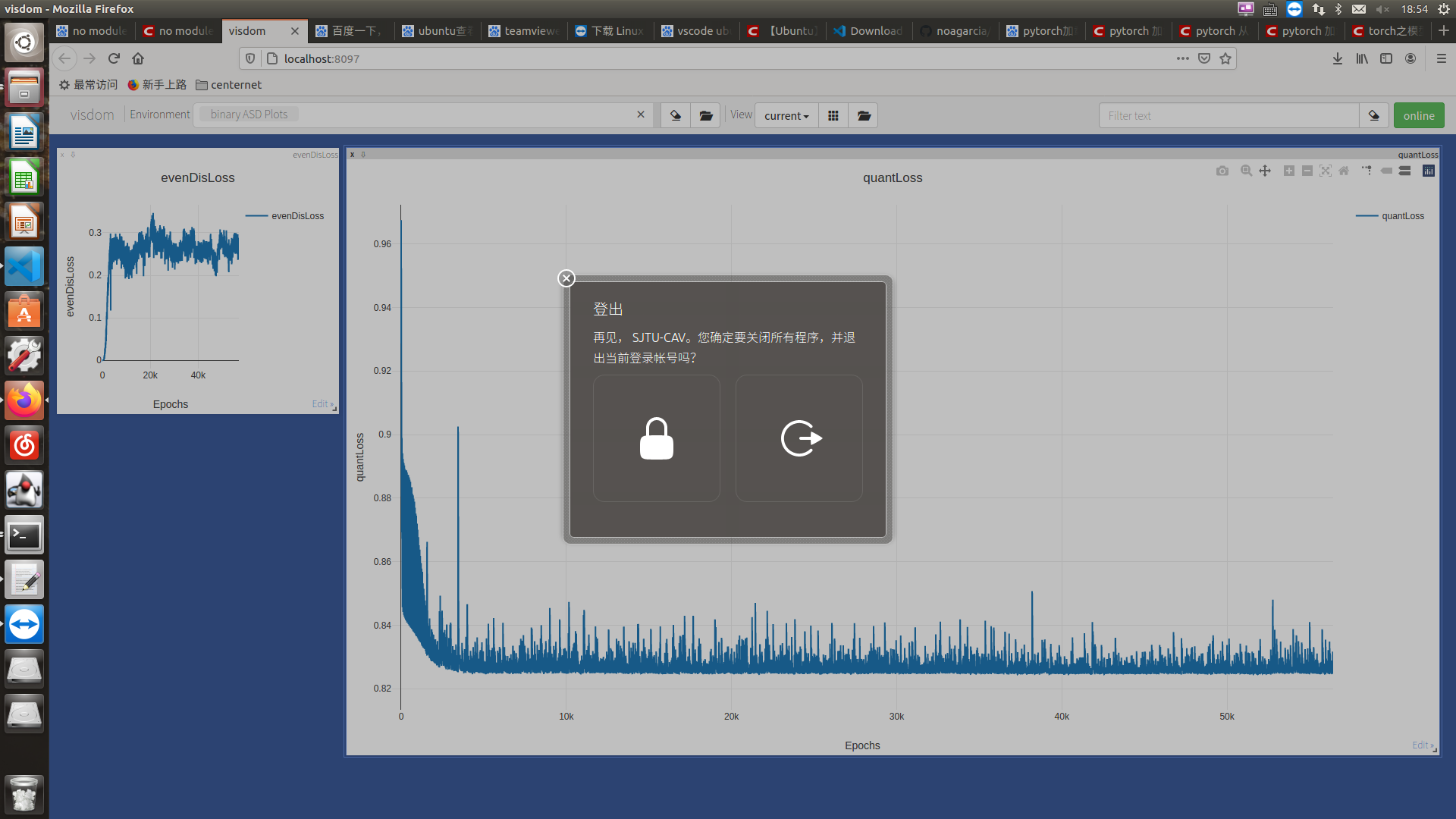
update W by minimizing the sum of T and Q



1. 将triplet loss和quantization loss交替训练，即每个batch size数据训练40次，前二十次训练triplet loss，后20次训练quantization loss。但是最终的效果也不好。



1. 不训练triplet loss，只训练quantization loss，损失降到0.8之后也不下降了（横坐标不是epoch，只是因为这个工具横坐标默认epoch，但是我觉得epoch打印的数据太少了，现在这个状态才一个epoch的20%）



1. 我想将马太原学长的网络的输出改为64维的描述子，然后后面利用k means聚类将1个浮点数映射到两个bit，可以表示四个状态，这样可以更多保留浮点数信息。

结果：生成的模型为1.3M，大概是128维的1/4，yosimite的test accuracy为3.4%，notredame的test accuracy为1%。

1. 将二进制用改进的sigmoid函数之后，二进制的效果比一般地二进制神经网络描述子都好。但是效果中规中矩，对比orb没有明显的优势，甚至在极端情况（大视角变化）下，比Orb效果还要差，这样一个没有特点的二进制描述符可能不足以撑起一个比较亮眼的论文。因此我们需要找一些比较好的优点，至少在某些极端情况下要超过orb，比如大视角变化，光照变化。

可以做的方法： 1）寻找相关的极端数据集，来训练一个好的模型（HPpatches有光照和视角变化大的图片序列得到的对应的patch对，可以最终用来训练我们的描述子）

2）改进自己的模型，改进loss function

3）在现有的数据集进行训练的时候加一些相应的噪声

对于sigmoid的偏移b的函数还是可以写进去一下

HPatches：有ref patch，easy patch（噪声小的），hard patch，每个patch是65\*65的像素

TartanAir: A Dataset to Push the Limits of Visual SLAM：一个检验光照和视角剧烈变化对于视觉SLAM的影响的数据集，跟kitti差不多，都是整张图片，对于我这个网络来说只能用来进行与orb的对比试验，不能用来做网络训练

1. 与其他二进制方法区别。代价函数那一块得提，
2. 加了sigmoid对于原来网络输出的尺度问题能否解决得更好？如果更好（最好比ASDNet），这个文章就比较完整，不仅能解决float，还能解决二进制。在解决尺度问题更好的同时还能增加适用性（float，二进制，内存，速度），也可以继续深入尺度理论的研究。
3. 也可以不用解决得更好，也可以虽然没有好很多，也有很多好的（好个两三个·），但是我的能够有更好的适用性，能够同时适用于float和二进制。
4. 和当前的其他二进制方法进行比较，找出一些理论性的东西
5. Sigmoid的t为什么要阶段性训练，带来了什么好处。最好能解决二进制问题，解决尺度问题。
6. 如果工作量不够，可以看看loam和lego-loam怎么升华的，自己也可以用在TX2上。有时间的话可以把SLAM那块做好。三维重构也可以，AR，VR，或者专门解决回环问题（大角度和光照）。也可以解决高重复场景（这个可能是个悖论，可能是因为ASD在这个场景下的正确匹配很多，用RANSAC把错误的都干掉了，而ORB没有很多正确匹配）。
7. 看看这个二进制在图片检索等方面有没有什么优势
8. 二进制理论研究

**最后发现用加了温度系数的sigmoid函数效果很好，应该寻找论文，探寻一下里面的秘密**

1. 现在发现一个很尴尬的问题，国庆节时写的代码的损失函数写错了。量化方程写错了。我开始的想法是利用改进的sigmoid函数对ASDNet输出的浮点型向量输入到改进sigmoid函数中，将输出靠近0和1，达到二进制化的目的。当我们一步步增大T的时候，sigmoid函数越来越接近阶跃函数，能够更好的降低量化损失，但是又不存在梯度为0，无法反向传播的问题。但是如果一开始T很大的话，则导致sigmoid梯度饱和，训练很慢。因此我设定的T=5\*epoch+1，一步步增大T，最终达到一个较好的二进制效果。



|  |  |
| --- | --- |
|  |  |
| T=1 | T=80 |

但是当时损失函数写错了

我将ASDNet最后一层的Batchnorm2d注释掉了，因为这个会限制ASDNet的网络输出的x在[-0.1,0.1]左右，导致我的sigmoid\_t函数的作用区间很小，而且靠近0的这一小块区域梯度很大，没能达到降低量化损失的作用。（这里我的理解有错误，其实batchnorm2d是用来将数据进行归一化到均值为0，方差为1的区间，而且batchnorm2d是有一个比例参数和偏移参数可以学习，可以将均值和方差区间进行偏移和增大。）

但是我在写量化损失函数的时候。首先将图片输入ASDNet，得到浮点型向量（当时没有验证，其实去掉batchnorm之后范围只是变为[-0.2,0.2]左右）。然后输入sigmoid\_t函数中，变为(0,1)区间，这是本应该用0.5将这个区间的值换算成0和1，但是当时用0做阈值，这样所有的输出的二进制的结果都是1。但是最后的效果还可以，跟CVPR的差不多。

我现在把所有的Batchnorm都注释，将ReLU改为tanh，这样再ASDNet输出的x值就在(-1,1)之间，然后再输入到我的sigmoid\_t函数，再利用0.5将输出结果分为0和1，再计算量化损失。需要看结果怎么样，但是训练的不是很乐观（效果不好）

我在马太原学长的网络下直接进行二进制化测试fpr95，然后发现效果其实跟IEEE那篇论文差不多

不能再陷入学长这个二进制的死胡同里面了，必须重新看看大量的二进制和浮点型的深度学习描述子了，看看别人的方法，自己能不能做点改进

精读几篇好的文章，你看别人做了啥，有啥不足的

其实我们的这个二进制化与神经网络的参数的量化还不一样，不需要我们的量化值与原来的浮点数必须接近，才能达到相同的结果。只要两个值能够同时为0或者同时为1，就能够分类成功。所以我们可以将我们的浮点数向量的量化当成一个多维的二分类问题，然后计算分类损失。

我们的二进制结果只跟浮点型的符号有关，跟浮点型的数值无关，所以用量化损失，将浮点型强行与±1靠齐的意义不大，而是应该对于out\_a和out\_p的的值进行二值化，然后如果对应符号相同，则损失为0，如果对应符号不相同，则将其差的绝对值作为损失，或者选择小（或大）的一个值的绝对值作为损失项。但是可能绝对值的差的损失应该不会很大，因为如果选择两个距离最近的a和p，势必两者的符号是差不多的

我们也可以用一个预训练好的模型进行后续的量化