**课堂笔记**

1. **KNN（40%准确率）**
2. 总括

KNN是一个图像分类算法，它使用某种距离指标衡量两幅图像之间的相似度。当判断一个新的图像x的类别时，会将x和所有已知类别的图像比较相似度，使用最相似的k张图像进行投票，从而确定x的类别。K值此处是超参数。交叉验证

将训练集分成K份，对每一个超参数，循环选取K-1份用作训练集，1份用作验证集。此时，对于每一个超参数将会得到K个准确率，取其均值作为其准确率。

交叉验证使训练集中每个数据都有作为验证集的机会，在数据集较小时被使用。

1. 优劣势

优势：算法简单

劣势：训练时时间少，只需存储（存储空间大）即可；测试时耗时长；像素距离与感官距离有很大不同，因此准确率不高；图片分类由背景主导，而不是景物（背景所占像素空间较大）

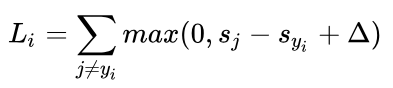
1. **线性分类器**
2. 总体算法

使用一个评分函数，将图像的像素值映射为各个分类类别的得分；使用一个损失函数，用来量化预测分类标签的得分与真实标签之间一致性。因此可将转化为一个最优化问题，在最优化过程中，将通过更新评分函数的参数来最小化损失函数值。

1. 线性分类器的映射函数



1. 损失函数
2. 多分类SVM损失



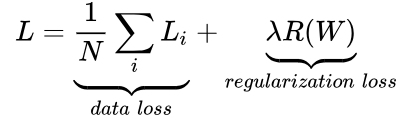
Max(0,-)称为折页损失

多类SVM“想要”正确类别的分类分数比其他不正确分类类别的分数要高，而且至少高出delta的边界值。

**Delta**: 大多数情况设为1.0。和正则化参数比重一样在损失函数中的数据损失和正则化损失之间做权衡。权重W本身缩小或者放大会使分类分值缩小或者放大，因此仅仅讨厌不同分类分值边界的具体值为1或者100是没有意义的，因为W本身就可以控制差值的变大和缩小。也就是说，真正的权衡是我们允许权重能够变大到何种程度（通过正则化强度lambda来控制）。

**正则化**：满足最小Li的W可能有很多(aW都是)，因此向某些特定的权重**W**添加一些偏好，对其他权重则不添加，以此来消除模糊性。我的理解是，一方面如果利用正则化，可以给权重加一些喜好，使最终得到的是符合喜好的结果，达到了在多个可能的函数中选择自己喜欢的函数的效果。另一方面通过正则化让某些权重值大一些；某些权重值小一点，而不是给权重更多的自由，可以用来防止过拟合。

L2惩罚倾向于更小更分散的权重向量，这就会鼓励分类器最终将所有维度上的特征都用起来，而不是强烈依赖其中少数几个维度。在后面的课程中可以看到，这一效果将会提升分类器的泛化能力，并避免*过拟合*。而b对输入并没有影响，因此不用正则化。最常用的正则化惩罚是L2范式，损失函数就变为下面的形式，



1. Softmax分类器

其损失函数使用交叉熵损失



从概率论的角度来理解，我们就是在最小化正确分类的负对数概率，这可以看做是在进行*最大似然估计；从信息论角度来看，交叉熵损失函数“想要”预测分布的所有概率密度都在正确分类上。*

1. **最优化**
2. **梯度计算**

利用有限差值计算梯度（近似数值法）或者微分分析计算（求微分）。在实际中使用分析梯度法，然后使用梯度检查来检查其实现正确与否，其本质就是将分析梯度法的结果与数值梯度法的计算结果对比。

1. **梯度下降**

Q:为什么梯度下降求最小值，梯度上升求最大值

E：梯度是变化最快的方向w-=a\*delt，取的是负方向，所以是下降的；w+=a\*delt取的是正方向，所以是上升最快的。

1. **小批量梯度下降**

对于训练数据很大的情况下，每次梯度更新都在整个数据集上计算量太大。可以选用小批量。之所以可行，是因为训练集中的数据都是相关的。计算整个数据集梯度的结果和计算数据集中一部分代表性数据集的梯度结果是差不多的。Batch大小用32，64，128等2的指数。

1. **随机梯度下降**

当每个batch只有一个样本的时候。

1. **反向传播**

**注意点：**

1. 在前向传播时对传播变量进行缓存
2. 不同分支的梯度要相加。前向的时候有多个分支，反向的时候多个分支要累加到分叉口。前向时多个分支合并，反向时copy每一份到每个分支。
3. **神经网络**
4. 非线性函数的引入（激活层）

如果不引用非线性函数，用再多的参数W1,W2,W3,最后都会变成某个W\*x(W=W1\*W2\*W3)，仍然是个线性函数，本质上并没有改变。

关于sigmoid的not zero-centered原因：sigmoid输出值即该神经元1输出x全为正，并传入神经元2.反向传播到2的梯度delt假设为正，则w梯度为神经元1输出x\*simoid导数\*delt。此处simoid导数要么正要么负（激活函数单调，导数同向），delt为正，x为正，所以w全为正，导致w要么同增要么同减，会zigzag的收敛。如果目标不在二四象限，就会收敛缓慢。其他如tanh一般就不会这样，因为它的输出是0为中心的。

**激活函数的选择**：用ReLU非线性函数。注意设置好学习率，或许可以监控你的网络中死亡的神经元占的比例。如果单元死亡问题困扰你，就试试Leaky ReLU或者Maxout，不要再用sigmoid了。也可以试试tanh，但是其效果应该不如ReLU或者Maxout

1. 神经元

一个单独的神经元可以用来实现一个二分类分类器，比如二分类的Softmax或者SVM分类器。

1. 网络结构选择

拥有至少一个隐层的神经网络是一个通用的近似器。一个隐层就能近似任何函数，那为什么还要构建更多层来将网络做得更深？答案是：虽然一个2层网络在数学理论上能完美地近似所有连续函数，但在实际操作中效果相对较差。虽然在理论上深层网络（使用了多个隐层）和单层网络的表达能力是一样的，但是就实践经验而言，深度网络效果比单层网络好。卷积神经网络的情况却不同，在卷积神经网络中，对于一个良好的识别系统来说，深度是一个极端重要的因素（比如数十(以10为量级)个可学习的层）。对于该现象的一种解释观点是：因为图像拥有层次化结构（比如脸是由眼睛等组成，眼睛又是由边缘组成），所以多层处理对于这种数据就有直观意义。

1. L1和L2正则化

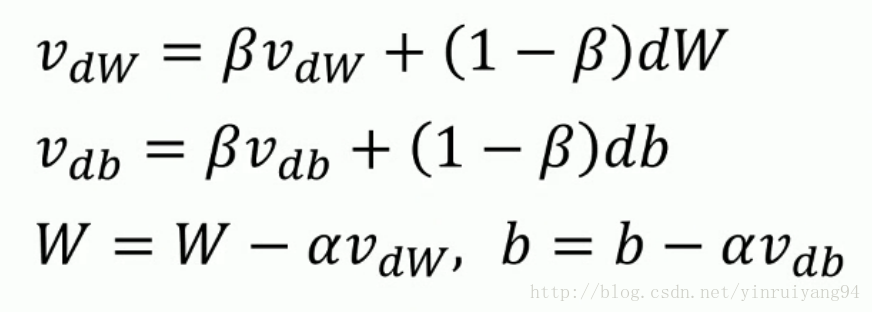
L1正则化有一个有趣的性质，它会让权重向量在最优化的过程中变得稀疏（即非常接近0）。L2正则化可以直观理解为它对于大数值的权重向量进行严厉惩罚，倾向于更加分散的权重向量。相较L1正则化，L2正则化中的权重向量大多是分散的小数字

1. **Batch Normalization为什么能有泛化效果**

BN后，min-batch中所有样本被关联在一起了，不会出现某个样本对结果产生很大影响的情况，因此有泛化效果。When training with Batch Normalization, a training example is seen in conjunction with other examples in the mini-batch, and the training network no longer producing deterministic values for a given training example. In our experiments, we found this effect to be advantageous to the generalization of the network.

6、随机梯度下降及各种更新方法

1）动量更新β是可以自行设置的超参数，一般情况下默认为0.9



可以看到v是之前的梯度的加权叠加，因此作出当前步长的决定不仅和当前的梯度相关，还和之前一段时间梯度相关。

举个例子，如果你站在一个地方不动，让你立刻向后转齐步走，你可以迅速向后转然后就向相反的方向走了起来，批梯度下降和随机梯度下降就是这样，某一时刻的梯度只与这一时刻有关，改变方向可以做到立刻就变。而如果你正在按照某个速度向前跑，再让你立刻向后转，可以想象得到吧，此时你无法立刻将速度降为0然后改变方向，你由于之前的速度的作用，有可能会慢慢减速然后转一个弯。动量梯度下降是同理的，每一次梯度下降都会有一个之前的速度的作用，如果我这次的方向与之前相同，则会因为之前的速度继续加速；如果这次的方向与之前相反，则会由于之前存在速度的作用不会产生一个急转弯，而是尽量把路线向一条直线拉过去。

这就解决了文中第一个图的那个在普通梯度下降中存在的下降路线折来折去浪费时间的问题。

2）Nesterov动量

4、**逐参数适应学习率方法**

1）Adagrad算法

随着迭代次数的增加学习率降低，缺点是，在Adagrad算法中，不断有正数加到分母的cache变量中，步长就会逐渐衰减到0，最后完全停止学习。在深度学习中单调的学习率被证明通常过于激进且过早停止学习。

当我们在垂直方向上看到许多大的梯度，Adagrad就会衰减学习速率，使垂直方向的更新步长越来越小。在水平方向上的梯度是很小的，所以分母会变小，相比于垂直方向，水平方向更新更快。这就是对每个参数自适应不同的学习速率，针对不同梯度方向的补偿措施。

2）RMSprop

用一种很简单的方式修改了Adagrad方法，让它不那么激进，单调地降低了学习率。

3）Adam

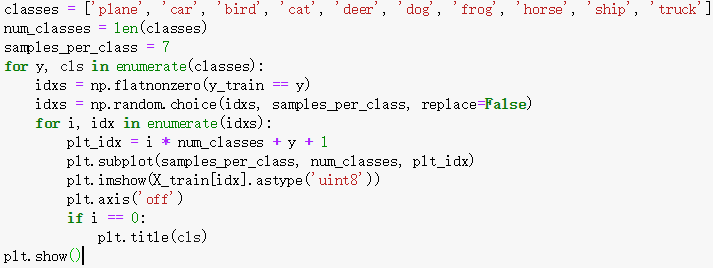
像是RMSProp的动量版

1. **dropout**

训练时每个神经元以概率p被激活。这样有N个神经元的网络有2^n中组合，原网络就相当于是2^n个网络的集成，这样就会有一定正则化的效果。要注意的是在测试的时候不会进行随机失活。但是对于两个隐层的输出都要乘以p，调整其数值范围。这一点非常重要，因为在测试时所有的神经元都能看见它们的输入，因此我们想要神经元的输出与训练时的预期输出期望是一致的。以p=0.5为例，在测试时神经元必须把它们的输出减半，这是因为在训练的时候它们的输出只有一半。为了理解这点，先假设有一个神经元x的输出，那么进行随机失活的时候，该神经元的输出就是px+(1-p)0，这是有1-p的概率神经元的输出为0。在测试时神经元总是激活的，就必须调整x\to px来保持同样的预期输出。

**Assignment1**

1. 可视化数据集中部分数据



1. 计算准确度

Accuracy=np.sum(y\_pred==y\_val)/y\_val.shape[0]

print ('k = %d, accuracy = %f' % (k, accuracy))

1. numpy广播

发生条件：当两个矩阵维度相同，或者任意一个维度是1。Eg.n\*1+n\*m=n\*m

应用场景：可以用广播和其它numpy函数实现向量化运算

1. numpy常用函数

argsort(a, axis=-1, kind='quicksort', order=None)Returns the indices that would sort an array，将a升序排序，返回排序后的index

l=np.bincount(x)#bincount统计x中各个元素（非负整数）出现的次数，返回次数列表。返回列表中下标i的值表示i在x中出现的次数

np.argmax(l)#返回l中最大值

X\_num=np.array\_split(X,num)#将X分为num份，X\_num[i]表示第i份

Np.sum(x,axis=1)按行求和

1. 模型常见写法

Class LinearClassifier(object):

def \_\_init\_\_(self):

self.W = None

def train(self, X, y, learning\_rate=1e-3, reg=1e-5, num\_iters=100,

batch\_size=200, verbose=False):

def predict(self, X):

def loss(self, X\_batch, y\_batch, reg):

1. continue#结束当前的循序，进行下一个数的循环,break是结束循环
2. {}表示字典，[]表示列表，()表示元组