**Lecture 1：概念**

1. 机器学习的基本框架：数据集，模型（函数），LOSS，优化算法

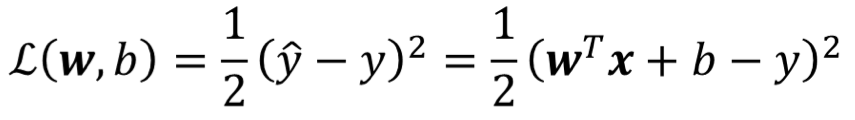
**Lecture 3：线性回归**

1. 一组数据，即输入与答案，称为一个样本（Sample）。
2. 线性回归选择的模型是线性模型：



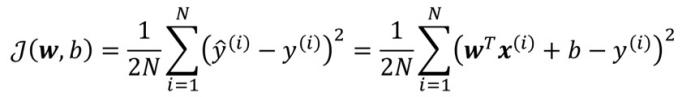
其中w和x是向量。

1. 何为线性？用人话说，线性在几何上是平的，不弯曲的；
2. 线性函数：形如y=的函数，特点是输出是输入特征的加权和。
3. 线性可分：用一条直线、一个平面、一个超平面可以直接分开的数据集。
4. 损失函数（Loss）：使用**均方差误差（Mean Square Error）**作为损失函数，它代表的是模型在某一个样本上的表现：



其中，平方是为了让损失不出现负数，我们只考虑模型做的好不好，有多不好，而不考虑这个好不好的方向；不用绝对值是因为求导方便，加个1/2也是为了求导方便。

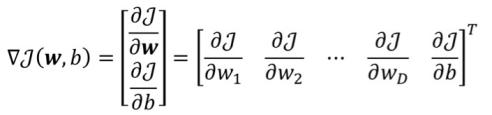
1. 代价函数（Cost）：把每一个样本的损失加起来，代表了模型整体的表现：



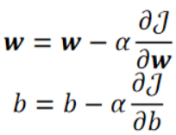
求和之后我们还乘了个1/N，是因为要知道的是模型的平均表现；其次，不是训练样本越多效果越好，也不是样本越多效果越差，但样本多了代价肯定相对更大，如果把样本多的和少的代价直接比较，显然不公平，因此要看平均表现。

损失函数（Loss）一定比代价函数（Cost）更加重要。

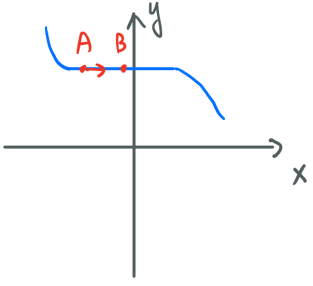
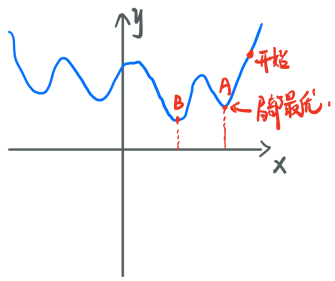
1. 梯度下降（Gradient Descent）：
2. 梯度：对所有参数求偏导之后合成的向量；
3. 梯度下降：梯度的反方向就是最快的优化方向。



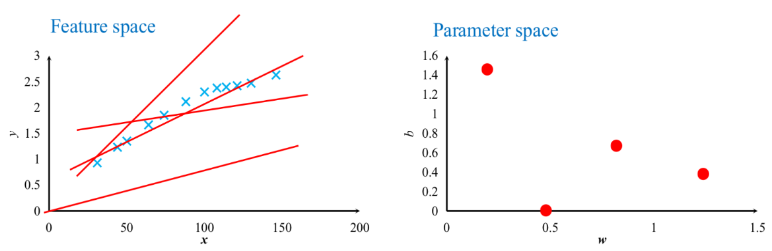
1. 优化：即优化参数w和b，直接减即可，学习率是一个超参数，需要人为控制。



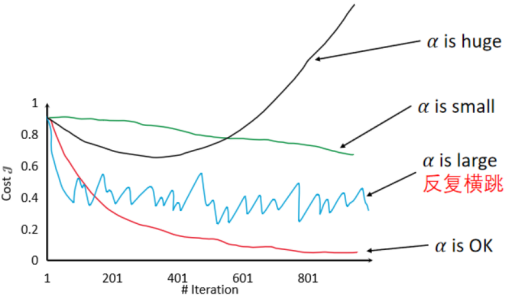
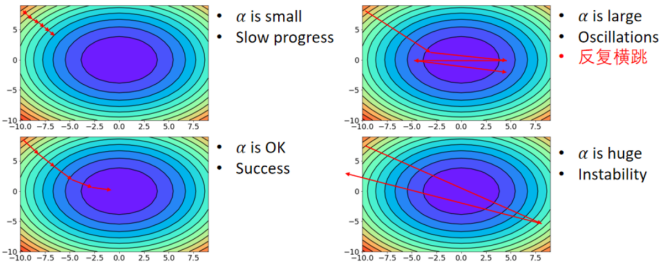
1. 梯度下降存在的问题：可能陷入局部最优（Local Minimal），不能保证找到最优解。



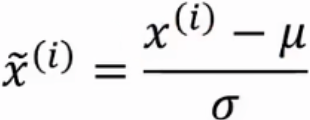
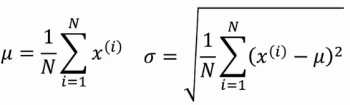
1. 两个空间：特征空间和参数空间
2. 参数空间就是把特征空间上每一条线（即每一个模型）的参数（w、b）画出来

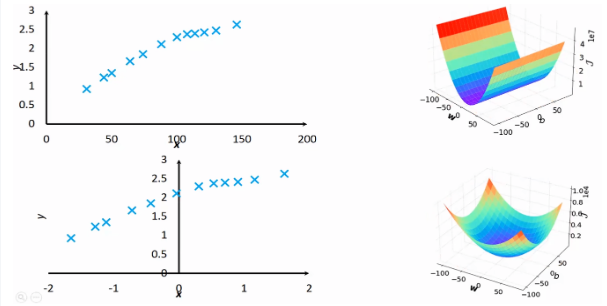


1. 学习率的影响
2. 学习率比较小的时候，学习过程很慢；
3. 学习率适中，效果就比较好；
4. 学习率较大，在最优结果之间反复横跳；
5. 学习率过大，非常不稳定，模型不可用。



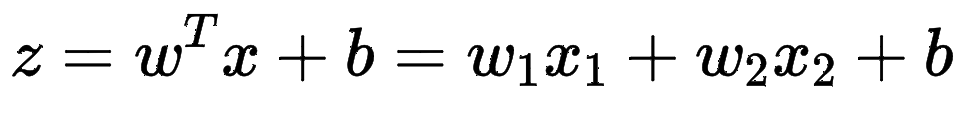
1. 归一化（Normalization）
2. 如果某个参数对结果的影响远远大于其它参数，这样是很难训练、很难调整超参数的。这种问题被称为尺度问题。
3. 归一化就是让所有参数的尺度相同，计算方式是每个样本减去样本均值再除方差。
4. 减去样本均值是为了让数据均匀分散在0的左右，除方差是为了把数据聚集在-1到1，整体跨度缩小。



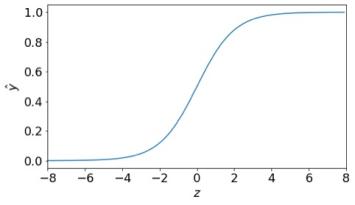
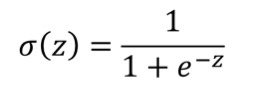


**Lecture 4：逻辑回归**

1. 逻辑回归用来做二分类。
2. 逻辑回归的样本的y是离散的。而线性回归的是连续的。
3. 我们选择线性模型来进行分类，即



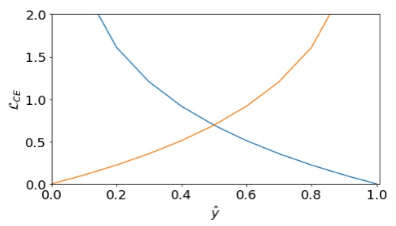
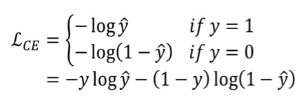
1. 对于最后分出来的线/面/超平面，我们不关心斜率，只关心能否分开。
2. 上面的模型得到的只是z而不是y hat，因为z得到的数范围为R，这不是我们想要的，我们想要的不是0就是1，因为二分类，因此我们需要通过**Sigmoid激活函数**来让输出在0到1之间：



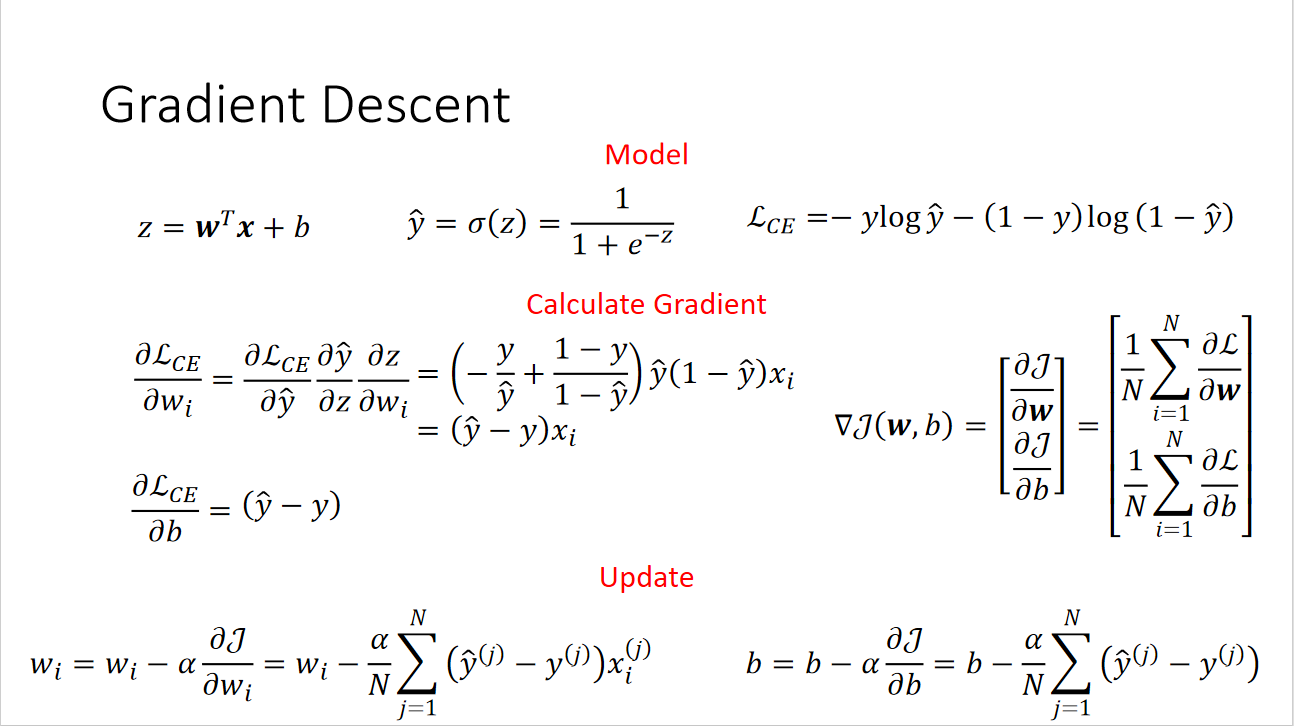
所以最终我们的模型变为：



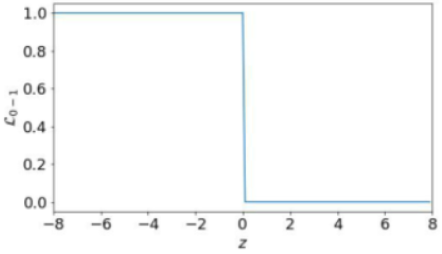
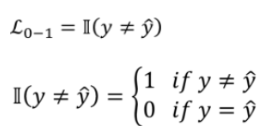
1. **交叉熵损失函数（Cross-Entropy Loss）**，我们选择它来作为逻辑回归的损失函数。
2. 交叉熵损失函数是专门用于分类任务的损失函数；
3. 在逻辑回归中，只需要做二分类，所以只需要使用二元交叉熵损失函数即可；



1. 梯度下降优化：

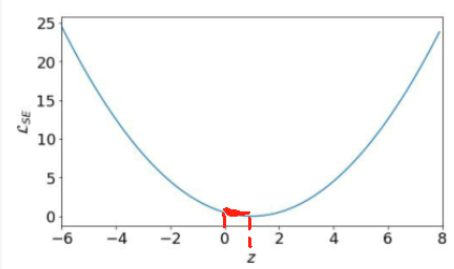


1. 损失函数选择的思考：
2. 为什么不使用0-1损失？



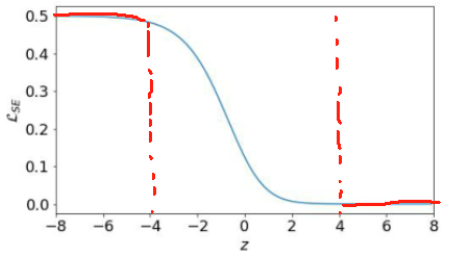
**导数处处为0！失去了梯度，即失去了学习的方向。**

1. 为什么不使用均方差损失（SE）？



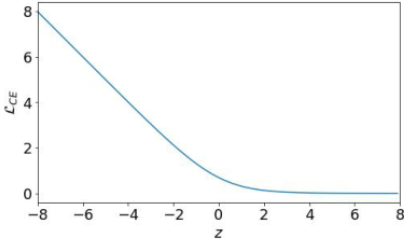
**优化空间非常小**，只能在[0,1]这个区间优化，因此优化程序需要拥有非常大的精度，不是用不了，只是要求太高，不用。

1. 为什么不用Sigmoid+均方差（SE）？



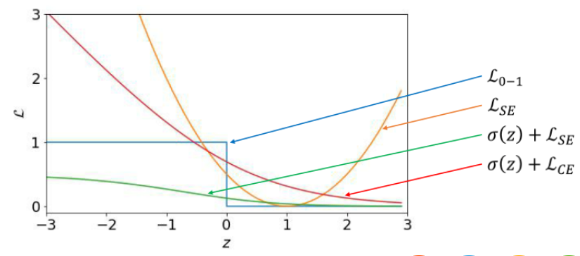
**Z<-4的部分斜率几乎为0，没办法进行梯度下降**，即“又差又学不动”。

1. 最后，选用Sigmoid+交叉熵损失函数



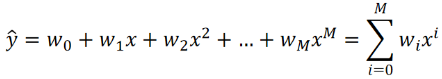
前三个的问题都得到了解决。

总结一下：



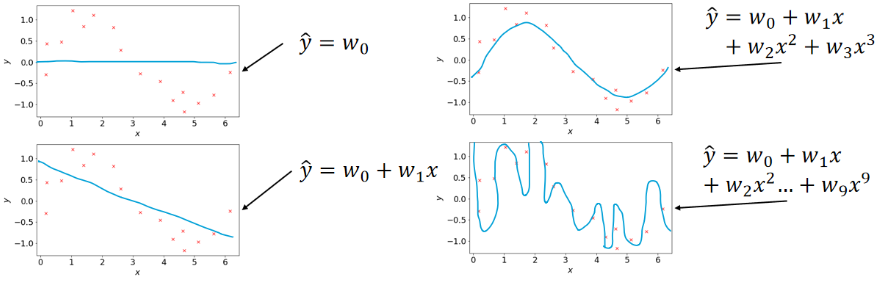
**Lecture 5：模型选择**

1. 多项式函数（Polynomial Function）可以用来做**回归预测**，值得注意的是，在多项式函数中，维度永远为1（D=1），即只有一个特征，可以看到式子里x都是一个x，次方不同而已。

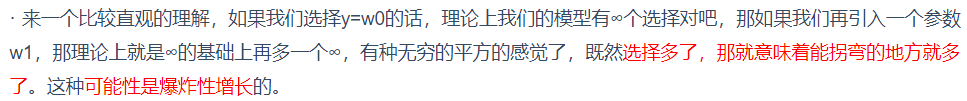


1. 欠拟合与过拟合

用多项式函数拟合点，得到下面四种结果



1. 引入的参数越来越多时，模型拐的越厉害？



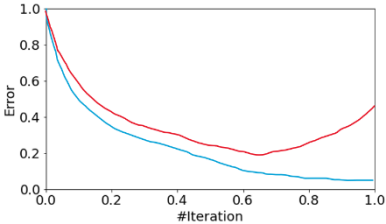
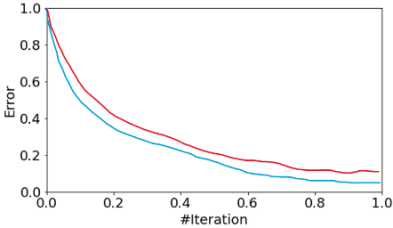
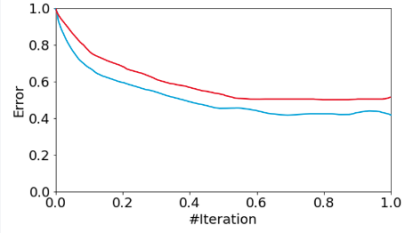
1. 过拟合（Over-fitting）：右下角的图就属于过拟合，它能力太强，可能性太多了，能把每个点都拟合上，这样对新来的点进行猜测，猜对的可能性就很小了。

解决方案：**换更简单、参数少的模型；继续采样，增大样本量；正则化（Regularization）**

1. 欠拟合（Under-fitting）：左上角的图就属于欠拟合，与过拟合相对，能力很差。

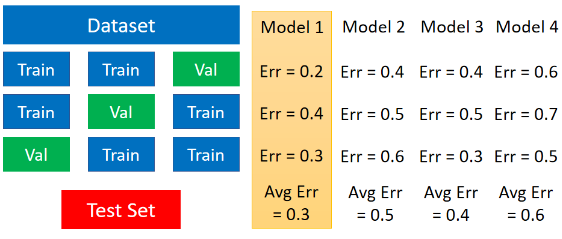
解决方案：**换更复杂、参数多的模型；增加维度、特征。**

1. 如何评估模型好坏？使用验证集。



三张图分别为欠拟合、刚刚好、过拟合。

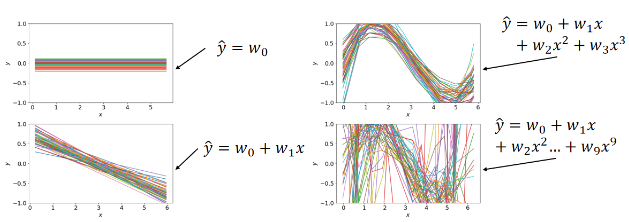
1. 训练集的表现肯定会比验证集上的好。
2. **N折交叉验证（N-fold Cross Validation）**：为了排除运气成分，选择真正好的模型。注意，只是尽可能排除运气成分，但不是100%排除。



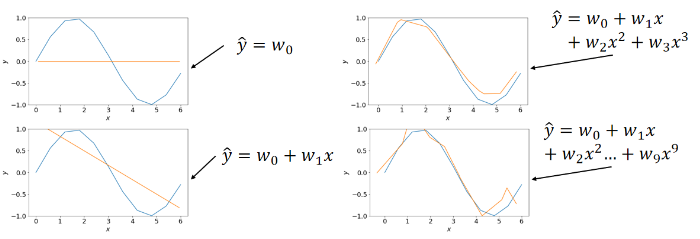
N折就是把数据集分为N份，然后每次取其中一份作为验证集，剩下的作为训练集，然后让模型用这三份训练集和验证集去跑，得到三个错误率，求个平均就是平均错误率，最后挑平均错误率最低的模型。

一般来说用10折，但是数据量过大的时候就用3或5折。

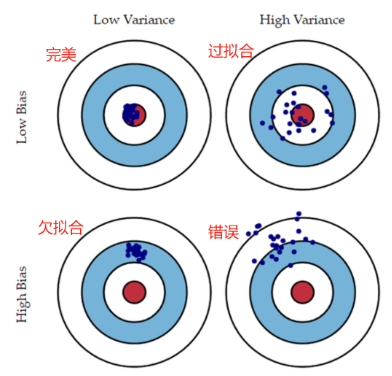
1. **方差（Variance）与偏差（Bias）**：在使用N折交叉验证进行N次训练之后，我们可以算出这个模型的方差（聚拢程度）和偏差（离正确答案有多近）。
2. 使用N折进行多次训练之后，模型表现出越来越混乱，所以越集中越有可能欠拟合，越混乱越有可能过拟合：



1. 对N折后的结果求个平均可以发现，求了平均之后，比较接近正确答案，因为尽管每一次都很随机，但总归是围绕正确答案转的，只要**次数够多**，求平均之后就能接近正确答案！

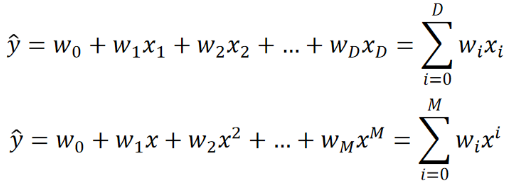


1. **用方差和偏差也可以判断欠拟合/过拟合**：



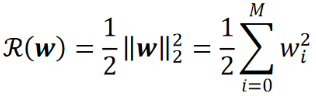
**Lecture 6：正则化**

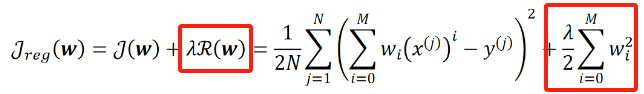
1. 正则化是降低过拟合的一种手段，但它不是万能的，在降低过拟合的时候会加剧欠拟合。
2. 如何判断一个模型是否复杂？计算**参数数量与特征数量的比值**。



第一个模型有D个参数D个特征，比值为1；第二个D个参数一个特征，比值为D，所以第二个比第一个更复杂，更容易过拟合。

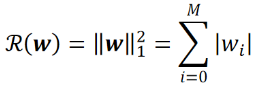
1. **L2正则化**：在原来的损失函数后加入一个对所有**参数值的平方**进行**求和**的项，参数数量越多这个项越大，以此作为惩罚，用一个拉姆达来控制这个惩罚的重要程度。





当拉姆达为0的时候，就是忽略正则项，模型保持过拟合；当拉姆达趋近无穷的时候，惩罚力度很大，模型就会趋向于**欠拟合**。

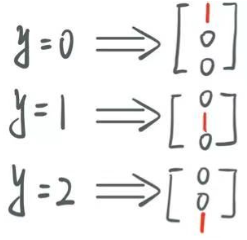
1. **L1正则化**：同样的在原来的损失函数后面加入一个对所有**参数值的绝对值**进行**求和**的项。



同样需要引入一个拉姆达来控制这个正则项的重要程度。

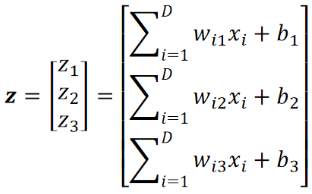
**Lecture 7：Softmax回归**

1. Softmax回归是用在做**多分类任务**的。
2. 独热向量（One-hot Vector）：既然有多个种类，那么标签肯定不止0和1，但如果使用2或者更大的数，就会造成一种暗示，因为算损失函数的时候，标签的值会影响损失大小，我们不希望如此，所以引入独热向量。



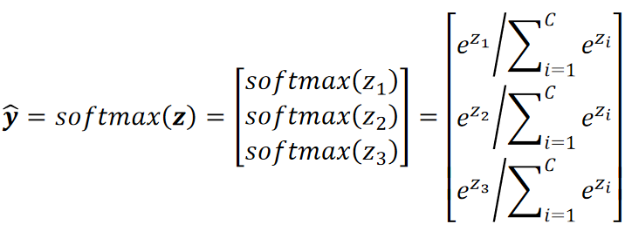
有N个类就创建长度为N的数组，在对应类别的地方设置为1，其它为0，这样就解决了暗示的问题。

1. Softmax回归中，我们使用的仍然是线性模型。但是，**有几个类别就用几个线性模型**。



一个模型计算一个类别的概率，比如第一个模型计算样本是第一类的概率。

算出来的是z还不是概率，需要过一个Softmax函数才能变为概率：



以**e**为底是为了**在保证概率不是负数的同时不改变三个z之间的大小关系**。

说白了，Softmax就干了两件事：

1. 把输出变为正数。
2. 把输出标准化在[0,1]，并且所有概率和为1。

可以把Softmax看作像Sigmoid那样的激活函数：



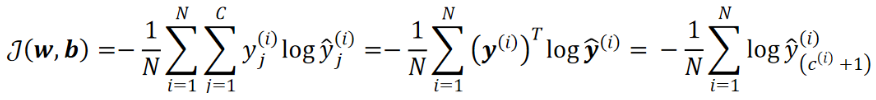
1. **交叉熵损失函数（Cross Entropy Loss）**：



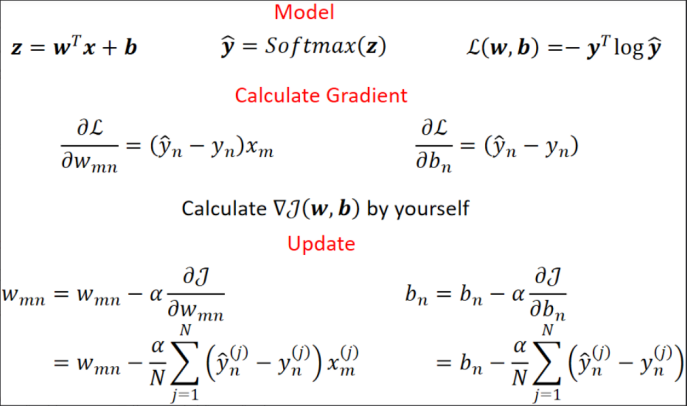
其实，因为我们把y都变为了独热向量，所以除了y=1对应那个y hat之外，其它y都是0，所以都不用算，所以就变成了只用算-log(y hat)：



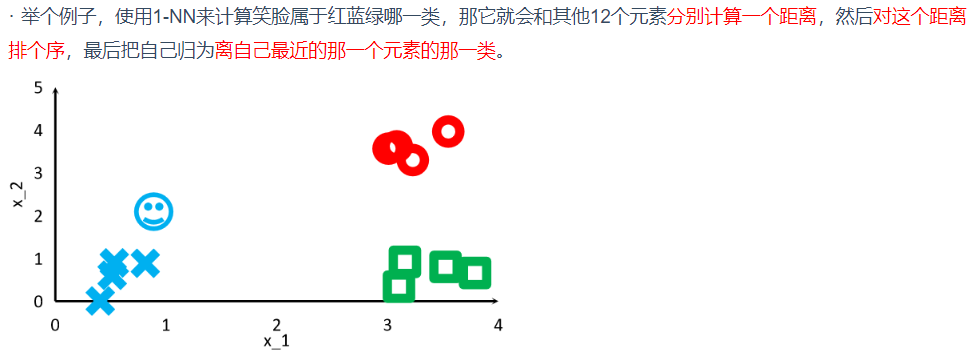
加起来就变成损失函数：



1. 梯度下降优化：



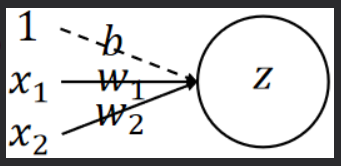
1. K-邻近（KNN）：也是用来做多分类的。K就是代表测试点考虑多少个邻居，最常用的是1-NN，即测试点选择离自己最近的点作为自己的类别。



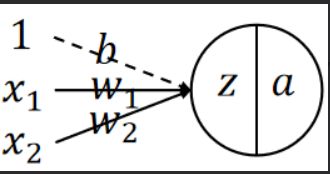
算法缺点很明显，就是**计算时间特别大**。

**Lecture 8：神经网络**

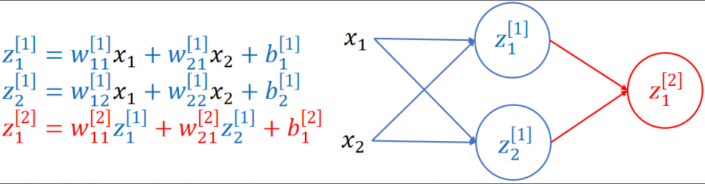
1. 神经网络本质上只是一个**特征提取器（Feature Extrator）**，只是对我们输入的特征做了强化，本质上没有改变其它任何东西！
2. 神经元（Neuron）：
3. 假设我们还是做多分类任务，那用到的模型还是线性模型，但我们可以用神经元来表示：



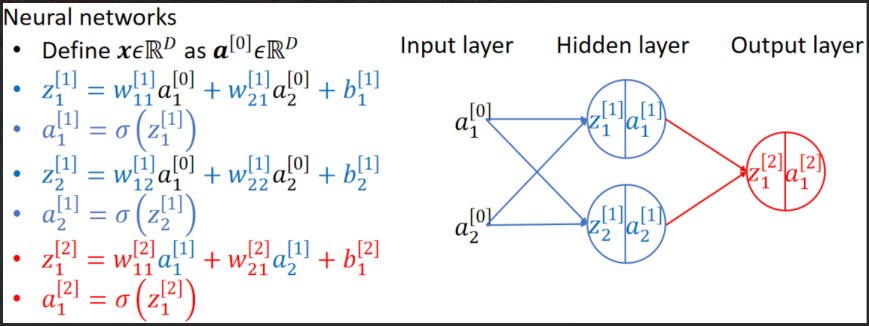
1. 使用Sigmoid作为激活函数来**加入非线性**：



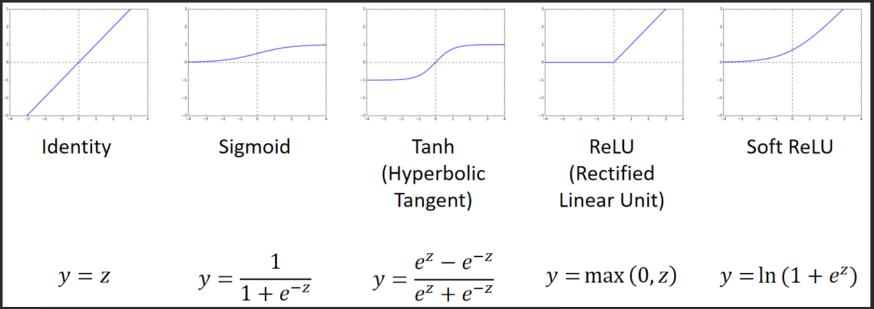
1. 神经网络：把神经元组合起来就是神经网络了，可以发现，神经网络的本质其实就是一个**多元复合函数**。



上面的还只是线性的神经网络，这不好，因为它本质上还只是一个单层的网络，把z1[1]和z2[1]代入z1[2]就可以发现中间层全部消失了，那它就只是一个简单的线性函数，所以要加上Sigmoid激活函数：

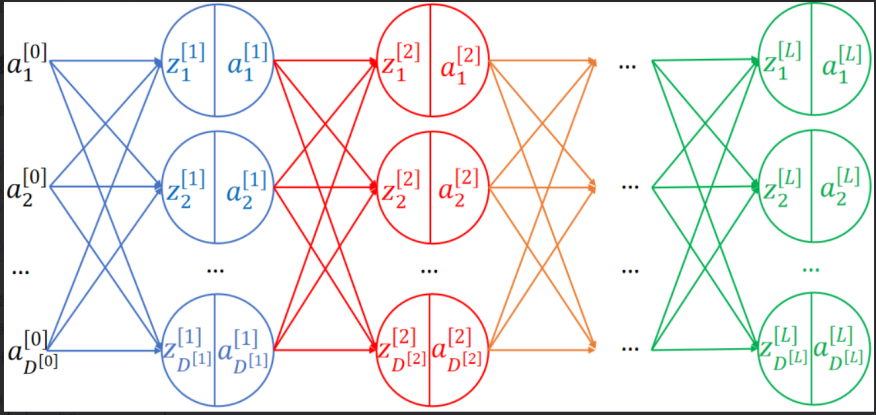


1. 激活函数（Activation Functions）：

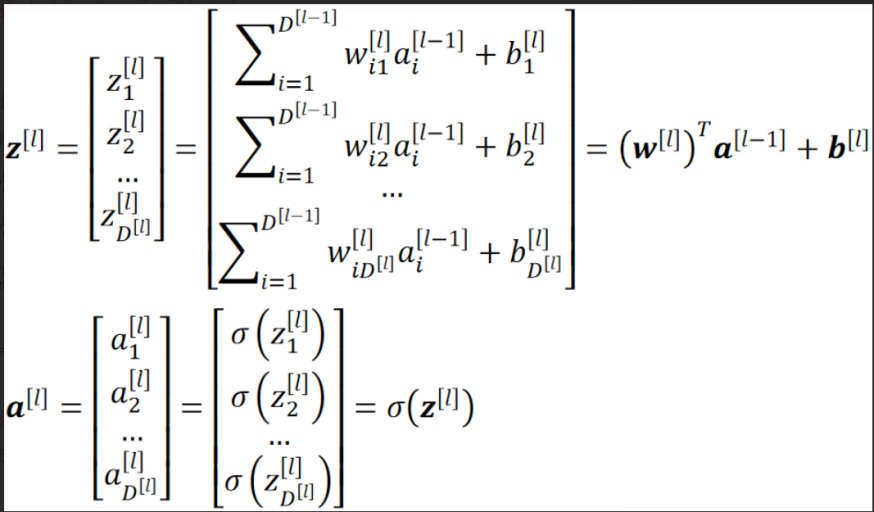


深度学习大多会选择**ReLU**，ReLU能解决Sigmoid导致的梯度消失或梯度爆炸的问题。

1. 全连接神经网络：有了最基础的神经网络架构之后，就能把它们连起来作为一个全连接神经网络了：

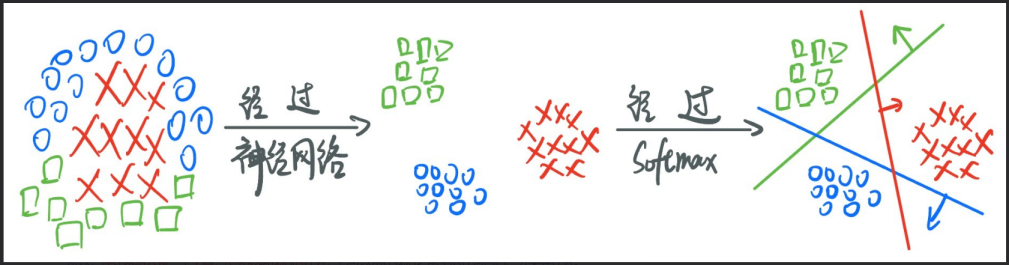


计算每一层z和a的方式如下：

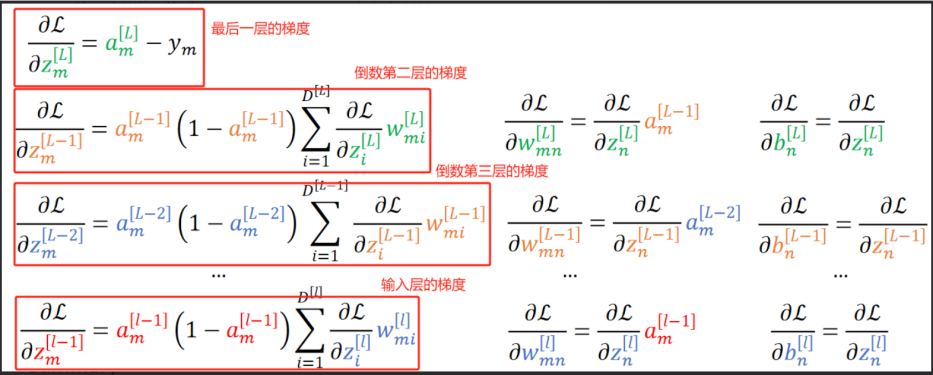


然后对于最后一层输出层，因为我们要做的是多分类，把最后一层的Sigmoid换成Softmax即可。

再次强调，神经网络只是用来**提取特征**的，不会对模型造成影响。



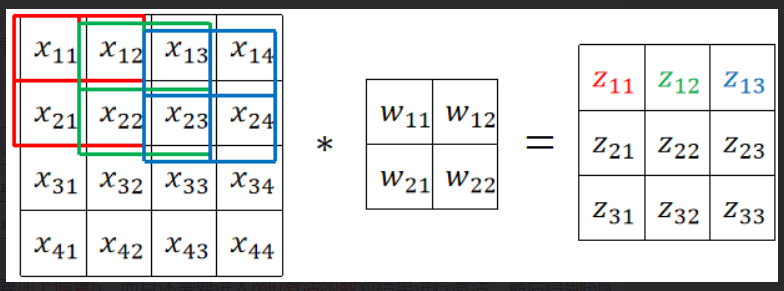
1. 损失函数和代价函数就因模型而异了，这里我们做的是多分类，所以用的还是交叉熵损失函数。
2. 反向传播：



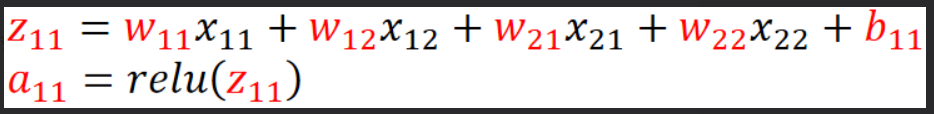
**Lecture 9：卷积神经网络**

1. 卷积神经网络是为了处理图片而生的，一般用于图片的**多分类**。
2. 样本：一张W\*H的图片，把像素拉直之后，共包含了W\*H个特征；如果我们考虑RGB三个通道，那特征数会翻3倍.
3. 为什么不直接使用传统神经网络？计算量过于庞大，一张800\*800的图片包含64W个特征，假设第一层神经元有10个，那就要计算64W\*10次，这只是第一层，属于深度学习的范畴，传统神经网络计算量太大了。
4. **卷积层**：卷积的作用就在于**大大减少了计算量**。首先定义输入图片大小Lin，然后定义卷积核大小K，然后定义卷积核步长S，那么就得到输出图像的大小为：





值得注意的是，卷积之后的结果需要加上一个偏置b再过一遍ReLU进行激活才是卷积层最后的输出。



1. **填充（Padding）**：原图片经过卷积之后会越来越小，这不是我们所希望的，因此我们在卷积操作之前进行补零操作：



这样卷积出来的结果就还是4\*4了。

1. 填充的位置可以是头部，也可以是尾部。
2. 如果要进行多次填充，就要**头尾轮流**填，否则会造成某个x被过度访问，某个x被过少访问的不平均现象。
3. 如果Padding次数过多，就会导致最后卷积核和一个全0矩阵相乘，这是没有意义的。
4. 一般来说最多也就是在图像外面包一层0，即P=2。

加入了Padding之后，卷积结果的计算公式就变为：



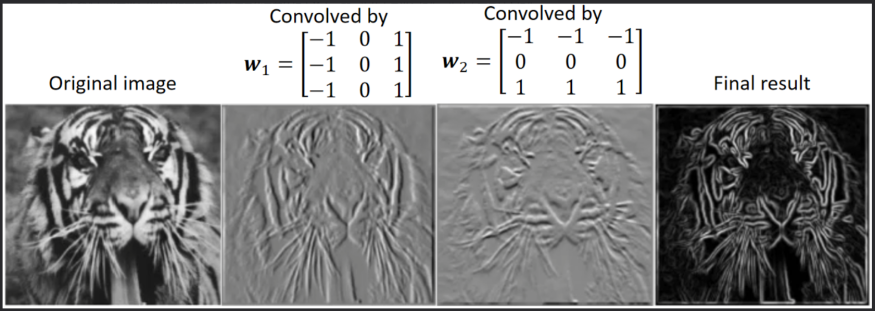
1. **浮点数计算量（FLOPs）**：这个数反应了模型的计算速度，越**小**代表越快。

比较全连接和卷积的FLOPs，假设输入输出图像都是5\*5，K=3，那么：

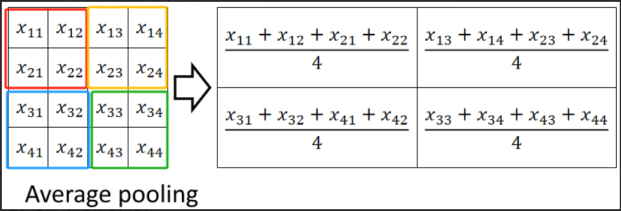
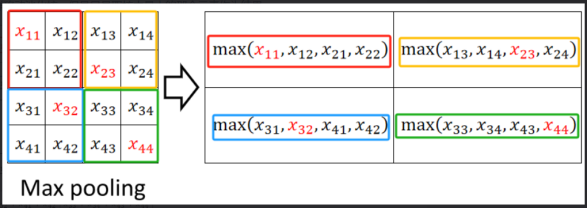
全连接:25个x和25个w相乘再相加得到一个z，即25\*25\*2=1250

卷积：卷积核走完整张图片得到一个z，即3\*3\*5\*5\*2=450

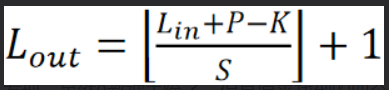
1. **为什么是卷积？**为什么卷积能提取图像特征？
2. 卷积核在CNN被发明之前就出现很久了，它被称为卷积滤波器（Conv Filter），他可以滤掉我们不需要的信息，只留下需要的。
3. 人为设计了很多卷积核，如Prewitt Kernel，他由两个核组成，分别对竖直和水平的边缘特征进行提取：



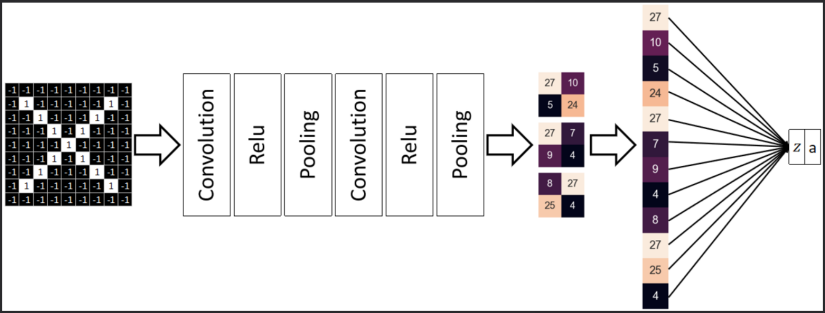
1. CNN的作用，只是帮我们学出来一个卷积核，然后通过这个卷积核提取出来的特征送入神经网络进行分类。
2. 使用多个卷积核其实提现的是分而治之的思维，每个卷积核都提取一部分特征。
3. **池化（Pooling**）：卷积运算的结果经过ReLU之后很多都变为0，这些0往后算都没意义了，因此我们希望把这些0丢掉一些以**节省计算量**。
4. 规定一个池化核的大小和步长，这两者一般都相等。
5. 池化操作一般是不做Padding的，因为就是为了丢掉0。
6. 进行最大池化（Max Pooling）和平均池化（Avg Pooling）：



1. 注意，**任何减少计算量的行为都会导致精度下降**，卷积也好池化也好都会。
2. 池化的结果也可以计算出来，一般P=0，K=S=2：

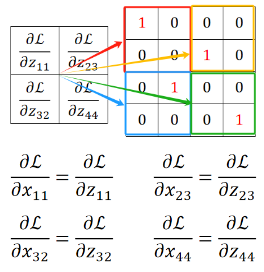


1. 前向传播总结：

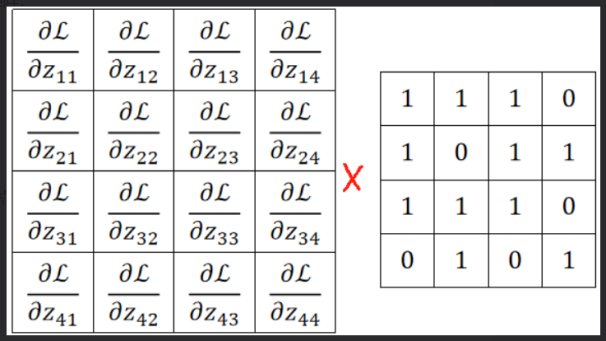


注意，无论是CNN还是NN，都只是**特征提取器**！真正完成分类任务的还是我们最后的Softmax。上述流程就是一个简单的LeNet。

1. 损失函数：多分类必然是交叉熵损失函数了。
2. 反向传播：
3. 池化层的反向传播：在进行正向传播的时候，要把最大值所在的位置记下来，然后把反向传播回来的四个dL/dz与最大值下标矩阵相乘就是dL/dx：

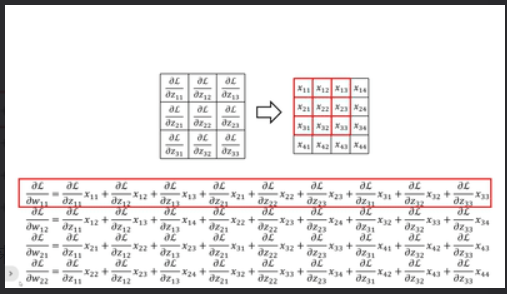


1. ReLU层的反向传播：在进行正向传播的时候，把是0和不是0的位置记录下来，然后拿池化层反向传播回来的dL/dz与记录矩阵相乘即可，乘完就是dL/dx：

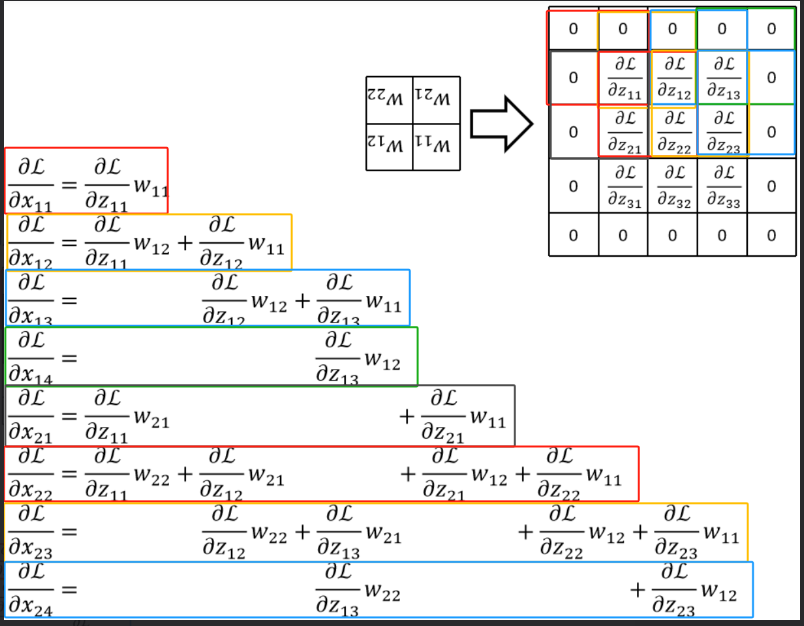


1. **卷积层的反向传播**：

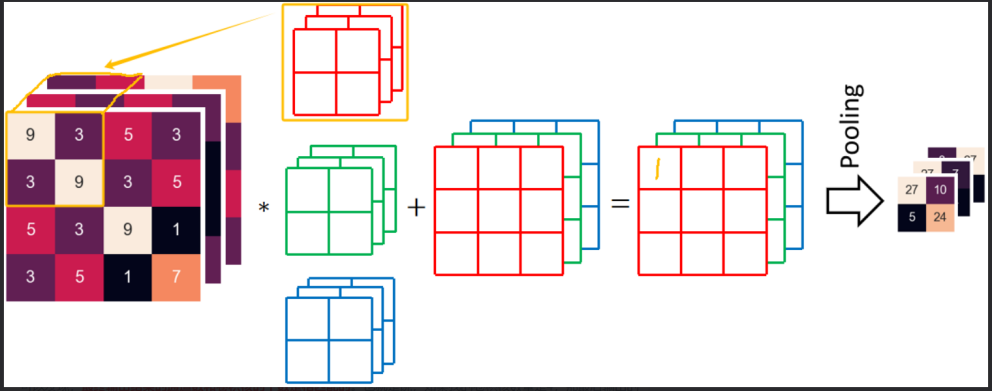
求dL/dw其实就是把反向传播回来的东西当做“卷积核”和原图做卷积，得到的就是对应的每一个dL/dw：



求dL/dx其实就是先把反向传播回来的东西补一圈0，然后把卷积核旋转180度，然后拿补0之后的矩阵和翻转后的卷积核乘，就是对应的一个个dL/dx：



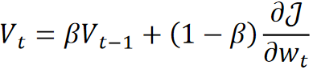
1. 多通道下的卷积和池化：卷积核的通道数要和输入图像的通道数相匹配；这一层的输入通道数由**上一层的卷积核数量**决定，也可以说，这一层输出结果的通道数与这一层的卷积核数量相匹配。



**Lecture 10：优化器**

1. 梯度下降容易卡在局部最优点，如鞍点（Saddle Point），平原（Plateau），这和初始化点有很大的关系；其次，因为不同方向上偏导大小不同的问题，会造成尺度问题，难以调控学习率。
2. 动量梯度下降GDM（GD with Momentum）：在更新梯度的时候考虑了上一次的动量和这一次的偏导：

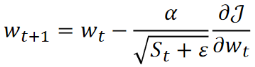


（也叫指数滑动平均Exponential Moving Average）

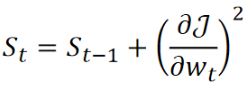
1. 使用超参数β来控制上一次动量的重要程度，一般β=0.9，初始动量V0=0。
2. GDM可以帮我们解决GD**陷入局部最优**的问题。
3. 随着越来越往后，前面的动量对当前的影响会越来越小。



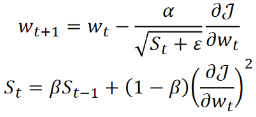
1. 自适应梯度AdaGrad（Adaptive Gradient）：用于解决**尺度不均衡**的问题。
2. 为什么不对每一个特征加一个超参数来控制它的大小？可以这么做，但是如今深度学习上亿参数不可能每一个都手动调，因此我们希望超参数也能自己学习。
3. AdaGrad的梯度更新公式如下：



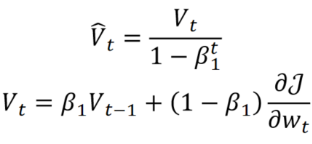
其中St为：



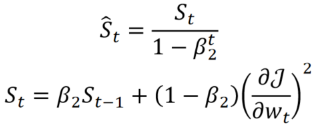
1. St由St-1决定，即上一次的梯度越大，那这一次的St就会很大，则这一次的梯度就会变小，达到了**动态控制学习率**的效果。
2. 缺点很明显，St会一直加下去，无限增大，那么到后面梯度就会非常非常小，学不动了。
3. 引入一个是为了**防止除0错误**，这个值很小，通常为10^-7.
4. 均方根传播RMSProp（Root Mean Square Propagation）：在AdaGrad的基础上加入了指数滑动平均，让St不会无限加下去：



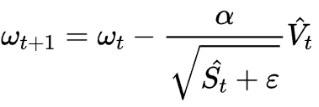
1. w的更新和AdaGrad是一样的，只是St的更新被限制了，这样前面的St对现在的St的影响就会越来越小。
2. 同样的引入了一个β来控制上一个St的重要程度。
3. 自适应矩预计Adam（Adaptive Moment Estimation）：本质上是GDM和RMSProp的结合。
4. 先通过GDM引入一个Vt：



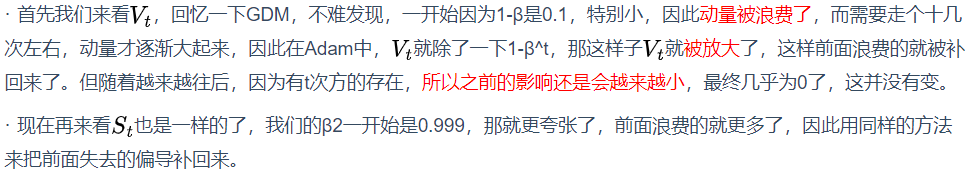
1. 再通过RMSProp引入St：



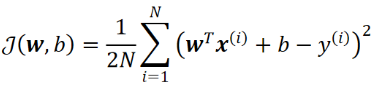
1. 最后把Vt和St揉在一起：



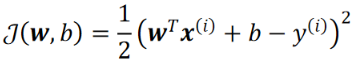
1. 这样Adam可以**同时解决**局部最优和尺度问题。
2. 通常
3. 为什么不直接使用Vt和St而是要对它们做一个处理呢？



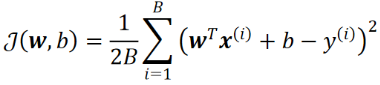
1. 其它的梯度下降方法
2. Full Batch：就是我们一直用的考虑所有样本的损失；



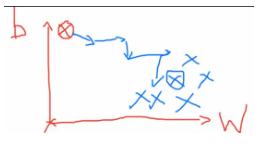
1. 随机梯度下降（Stochastic GD）：每次只挑任意一个样本的LOSS代表全体的，但每次挑的不一样；



1. Mini-Batch：只取样本的一部分的LOSS作为全体的，因此需要引入一个超参数Batch Size代表每次取多少个样本；假设我们样本数量为12，Batch Size=5，那多出来的两个可以选择扔掉，但更好的办法是**修改Batch Size让所有样本都被用上**！

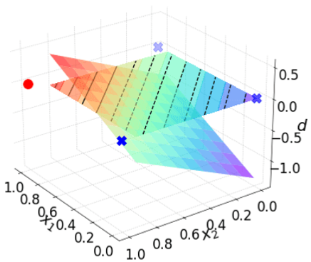


1. 随机和小批量梯度下降都能大幅度加快训练速度，但因为样本量少，每次计算最优点的时候都会有偏差，但问题不大，虽然过程歪歪扭扭，但最终肯定会向正确的方向前进。

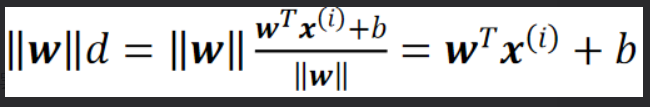
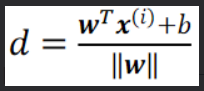


**Lecture 11：感知机**

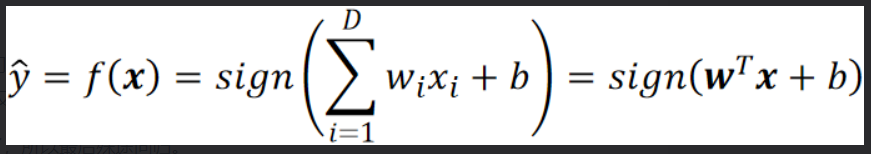
1. 感知机也是用于二分类的。
2. 在感知机里，标签变为**{-1，1}**而不是0、1.
3. 因为距离一致，所以不需要转换为独热向量。
4. 回归任务和分类任务的最大差别是：回归任务的标签y是**连续**的，分类任务的标签y是**离散**的。
5. 我们选用的依然是线性模型，我们要训练的模型其实是，因为本质上是z=wTx+b和z=0联立。
6. 我们给模型两边随便乘点东西，如拉姆达，是**不会**对模型的分类结果造成影响的，因为只改变了决策面的切的角度，但切上去的线不变。



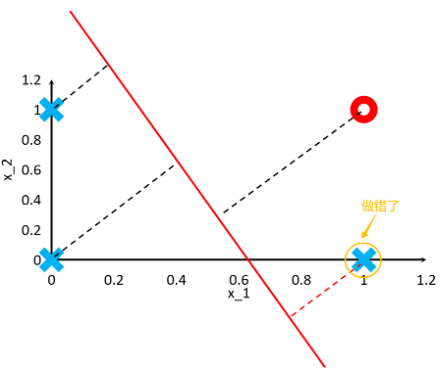
1. 我们计算每个点到决策面的距离，取符号以决定分到哪一组去；所以我们算出来的距离d不重要，只需要知道正负即可，所以我们可以给的乘一个||w||来简化计算：



最终的模型形式如下



1. 损失函数：所有做错了的点到决策面的距离的和。



① 但是这样求导很痛苦，其次很难界定一个点做没做错。

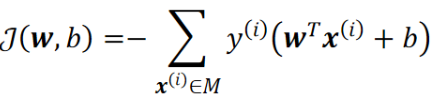
② 这个时候把标签改成{-1,1}的好处就来了，我们**把标签乘上距离**，如果大于0就肯定是做对了，小于0就是做错了：



1. 所以我们的损失函数就可以定义为：

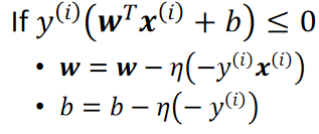


代价函数就是：



其中M是做错的所有点，所以要在最前面加一个负号，才能变正。

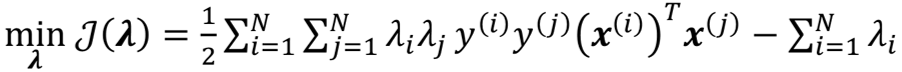
1. 优化：使用随机梯度下降。每次只挑一个样本作为loss，一直循环到没有错为止。



1. 感知机不能保证找到最优解，要使用SVM才能找到最优解。

**Lecture 12：支持向量机**

1. 支持向量机也是做二分类的，但是和逻辑回归、二分类不同的是，SVM想要找到**最优解**。
2. 根据互补松弛条件：
3. 对于松弛的限制条件，不会影响最优点的位置，λ必定为0，g(x)<0；
4. 对于紧致的限制条件，会把最优点强行拉过去，λ>0，g(x)=0；
5. 当样本**是**支持向量的时候，λ**>**0；
6. 当样本**不是**支持向量的时候，**λ=0**。
7. 计算公式：



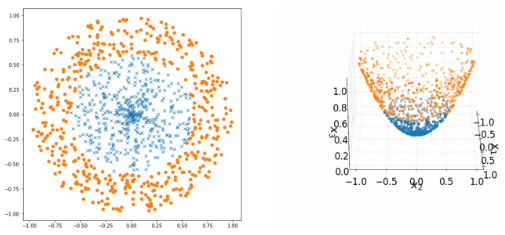
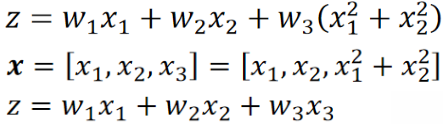








1. 核技巧（Kernel Trick）：SVM只能做**线性可分**的数据，**核技巧就是扩展维度，把低维的样本映射到高维去**，当一些样本的维度越高，越有可能线性可分，这样SVM就能用的上了。

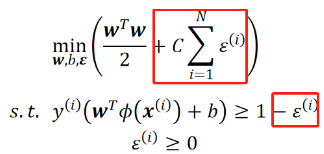


常用的核函数有：

径向基核函数（Radial Basis Function），用于映射到无穷维

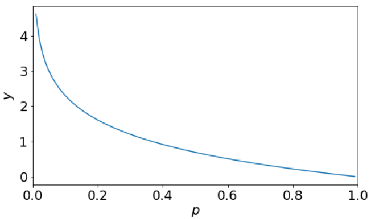
多项式核函数（Polynomial Kernel Function）：

1. 软间隔SVM：可以容忍一些噪声的存在的SVM，在限制条件上，我们放宽了一点，当然对应的需要加上惩罚，C是用来控制惩罚的力度的，C越大，越趋近于硬间隔SVM，C越小，越鼓励模型犯错。

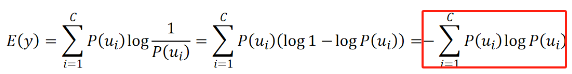


**Lecture 13：决策树**

1. 决策树适用于离散型数据。
2. ID3：
3. 构建决策树的时候，我们使用的是贪心启发式算法（Greedy Heuristic），即把所有指标都算一遍，选表现最好的那个指标。因此，构建出来的树**不一定是最好的**。
4. 惊讶程度（Surprise）：

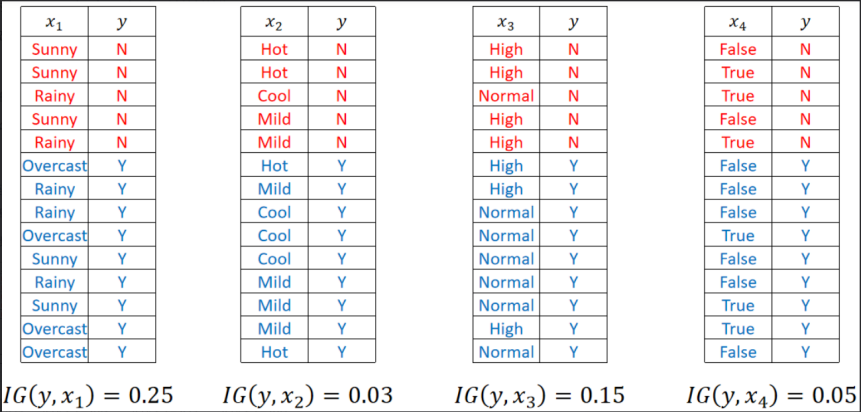


1. **熵（Entropy）**：表示惊讶程度的期望，用于衡量**某一类里面某一组**分的好不好。



1. **信息增益（Information Gain，IG）**：用于衡量**整一组**分的好不好。

计算IG：



计算之后，选择IG**最大的**作为这一层的分类指标即可。

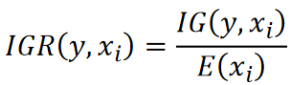
1. ID3的缺点：

·某一特征的取值可能性特别多，就会更倾向于选择这种特征，比如引入一个Index，14个都不一样。

·不支持连续的值。

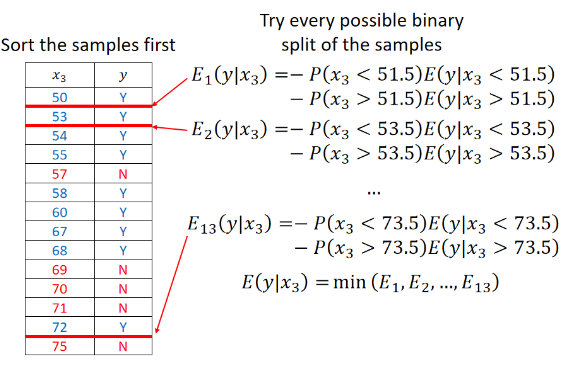
·不支持剪枝。

1. C4.5：
2. 引入信息增益比（Information Gain Ratio）：在IG的基础上，除了一下这个特征的熵。

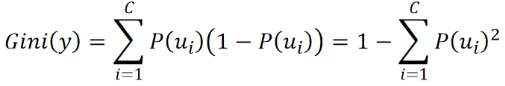


这样就避免了特征可能性多而倾向它的情况了。

1. C4.5支持连续值：给数据排序后，不断地做二元切分，因此数量不是1的那一类是可以**复用**的。



1. C4.5支持剪枝，预剪枝（边建边剪）；后剪枝（先把树建好，再剪枝）
2. C4.5一个节点可以有多个分支。
3. C4.5在连续值上不支持**回归**。
4. CART：
5. 基尼指数（Gini-Index）：另一种衡量混乱程度的指数，和熵类似。



1. 选择计算出来基尼指数**最小**的那一类作为指标。
2. 因为CART只分为两种，所以被分为“不是xx”的那一种里面还可以继续**复用**。
3. CART支持回归。

**Lecture 14：集成学习**

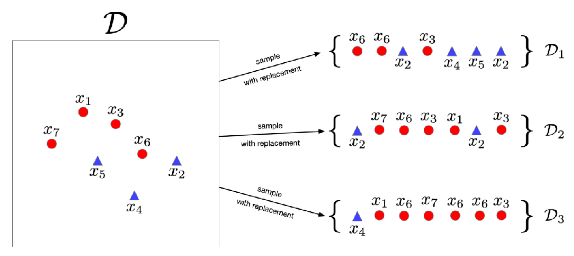
1. Bagging：

① 一种并联的方式。

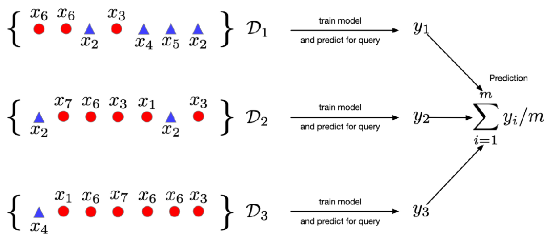
② 用来**降低高方差**，对偏差不起作用。

③ **工作流程**：

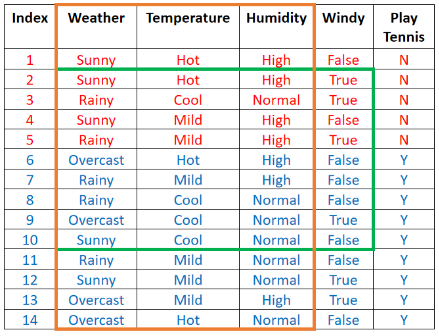
· **有放回地随机抽样（Bootstrap Sampling）**n组数据：



· 用这n组数据分别训练n个模型，然后把预测输出求个平均：

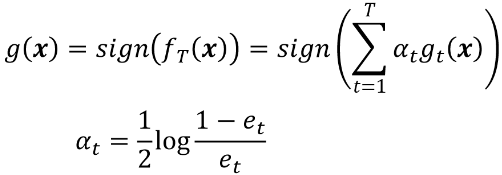


1. 随机森林：在决策树中做Bagging。先对行做Bootstrapping采样，再对列做，然后在选出来的框框里建树：



1. Boosting：
2. 是一种串联的学习形式；本质上是训练多个弱学习器；
3. Boosting用于**降低高偏差**，提高方差；
4. **AdaBoost**：

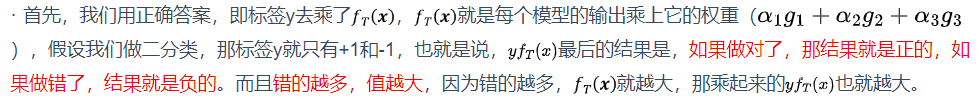
·把所有弱学习器的结果加起来，当然还要乘上每个学习器自己的权重。



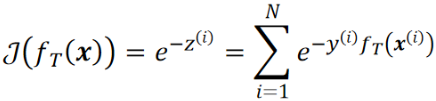
·权重可以根据错误率直接计算得到。

·损失函数如下

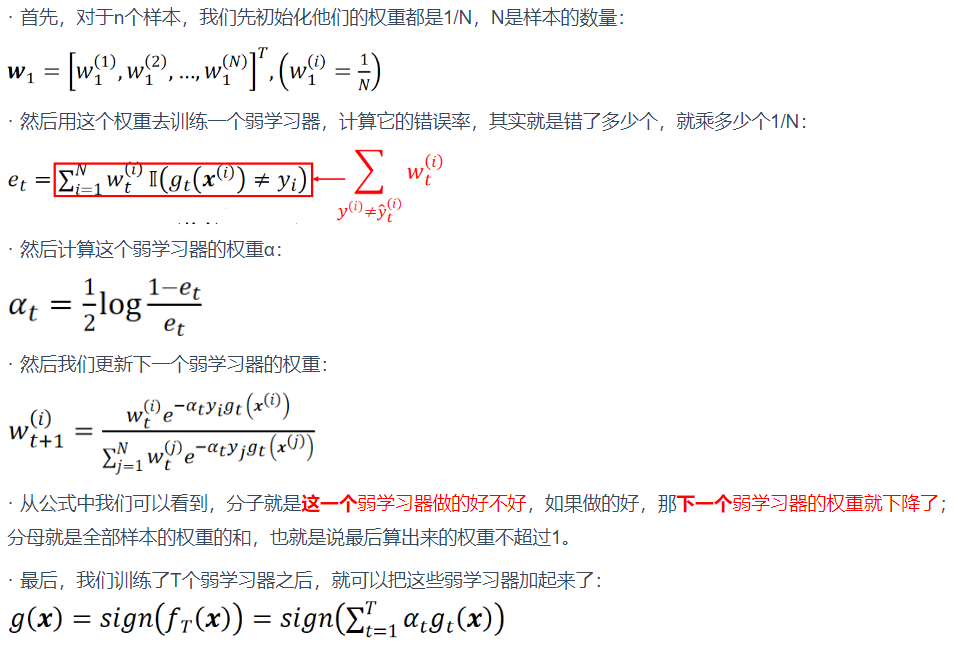




·代价函数如下，前N-1个模型的损失是固定的了，所以前N-1个模型的代价和就是第N个模型的权重。



·**工作流程**：



·**手算例子**：

