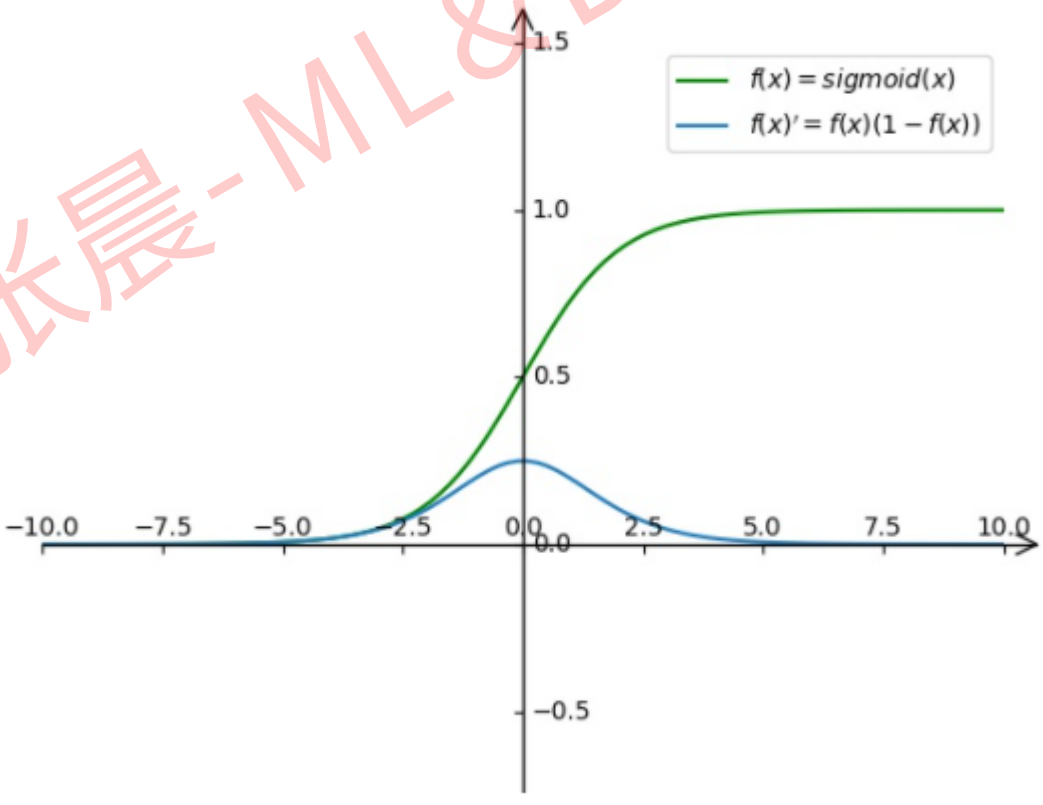
9. 深度学习基础知识★

1．激活函数及其导数

sigmoid：

$$f_1(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d}{dx} f_1(x) = f_1(x)(1 - f_1(x))$$



优点：

- 连续，且平滑便于求导

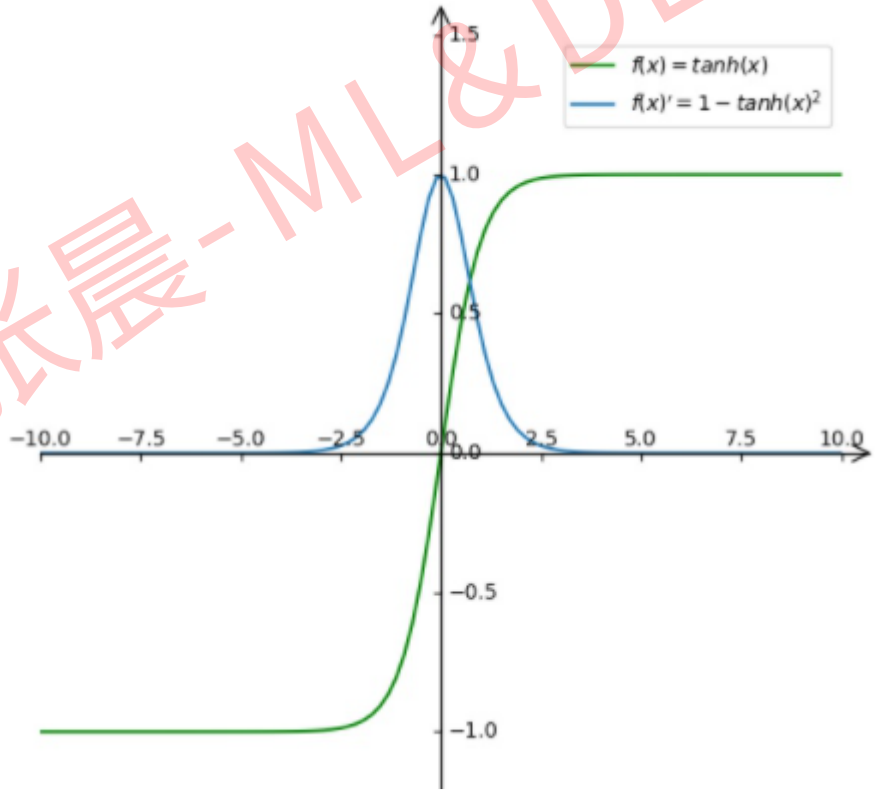
缺点：

- none-zero-centered（非零均值， Sigmoid 函数 的输出值恒大于0），会使训练出现 zig-zagging dynamics 现象，使得收敛速度变慢。
  https://blog.csdn.net/weixin_45268911/article/details/107321613 <https://blog.csdn.net/weixin_45268911/article/details/107321613>
- 梯度消失问题。 由于Sigmoid的导数总是小于1，所以当层数多了之后，会使回传的梯度越来越小，导致梯度消失问题。而且在前向传播的过程中，通过观察Sigmoid的函数图像，当 x 的值大于2 或者小于-2时，Sigmoid函数的输出趋于平滑，会使权重和偏置更新幅度非常小，导致学习缓慢甚至停滞。
- 计算量大。由于采用了幂计算。

tanh：

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{d}{dx} f_2(x) = 2f_1(2x) * 2(1 - f_1(2x)) = 1 - f_2^2(x)$$



优点：

- 它将输出值变换到了[-1,1]的范围内，这就解决了sigmoid函数以非0为中心的问题
- tanh 的导数的取值范围为(0, 1)，所以 tanh 的收敛速度会比 sigmoid 快

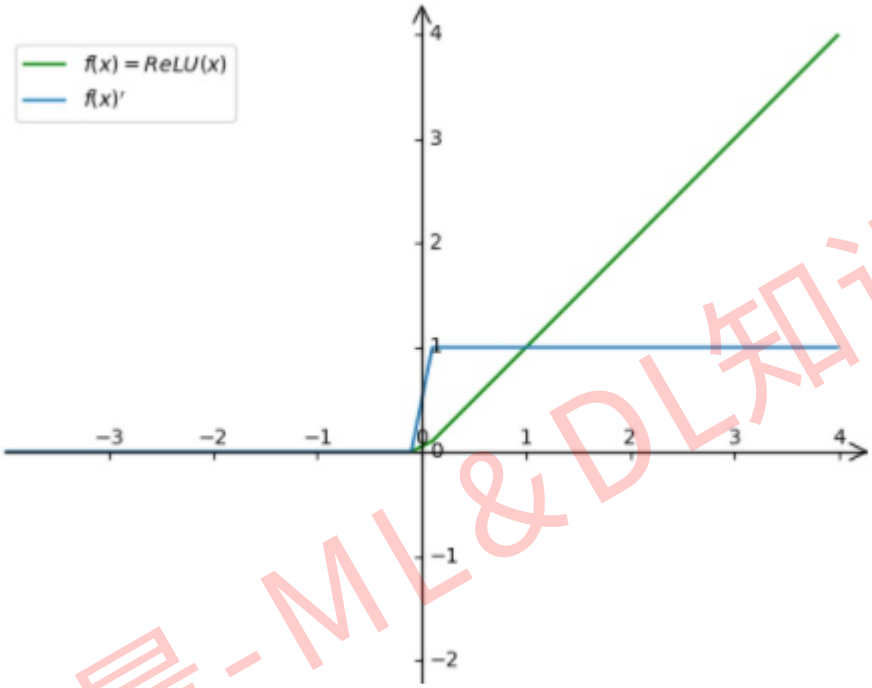
缺点：

- 梯度消失，
- 计算消耗大

relu：

$$f_3(x) = max(0, x)$$

$$\frac{d}{dx} f_3(x) = \begin{cases} 1, & x>0 \\ 0, & x \leq 0 \end{cases}$$



优点：

- x 大于0时，其导数恒为1，这样就不会存在梯度消失的问题
- 计算导数非常快，只需要判断 x 是大于0，还是小于0
- 收敛速度远远快于前面的 Sigmoid 和 Tanh函数

缺点：

- none-zero-centered
- 在0处不存在导数，tensorflow里面就默认relu在0点的导数是0。可以用log(1+ex)来近似，这个函数是连续的，它在0点的导数是0.5。
- Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。因为当x 小于等于0时输出恒为0，如果某个神经元的输出总是满足小于等于0的话，那么它将无法进入计算。有两个主要原因可能导致这种情况产生:
  - 非常不幸的参数初始化，这种情况比较少见
  - learning rate太高导致在训练过程中参数更新太大，不幸使网络进入这种状态。解决

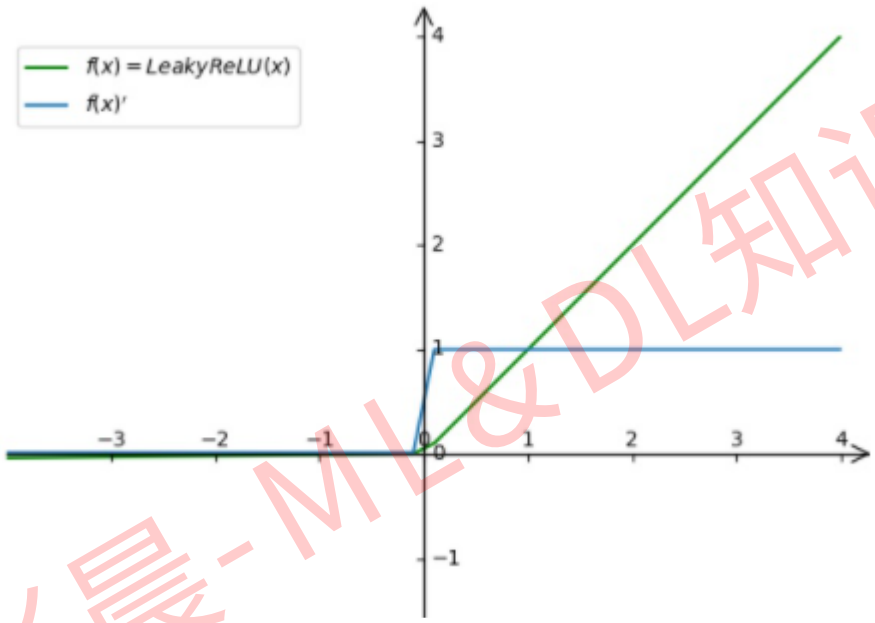
方法是可以采用 MSRA 初始化方法，以及避免将learning rate设置太大或使用adagrad等自动调节learning rate的算法。

leaky rulu：

$$f_4(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

其中，α是一个接近0的常数

$$\frac{d}{dx} f_4(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$



优点：

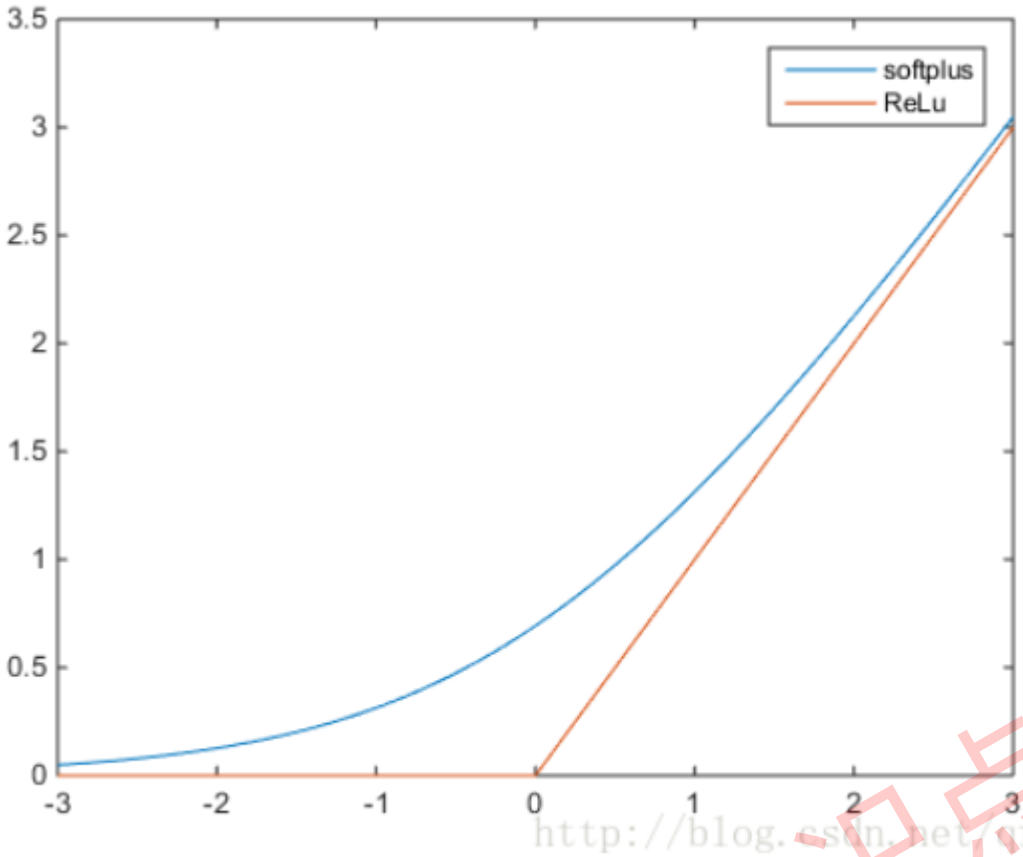
- x 大于0时，其导数恒为1，这样就不会存在梯度消失的问题
- 计算导数非常快，只需要判断 x 是大于0，还是小于0
- 收敛速度远远快于前面的 Sigmoid 和 Tanh函数
- 解决前面 Relu 提到的 Dead ReLUProblem 的问题，因为当 x 小于 0 时，其输出不再是 0.

缺点：

- none-zero-centered

softplus：

$$f(x) = \log_e(1 + e^x) \qquad f'(x) = \frac{1}{1 + e^{-x}}$$



优点：

- 解决了relu在0处不可导的情况
- 解决前面 Relu 提到的 Dead ReLUProblem 的问题，因为当 x 小于 0 时，其输出不再是 0.

缺点：

- none-zero-centered
- 计算量大，由于采用了幂计算

softmax：

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \qquad \forall j \in 1...N$$

$$D_j S_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_j S_i & i \neq j \end{cases}$$

优点：

- softmax 可以用于多分类问题

缺点：

- 计算量大，由于采用了幂计算

maxout：

$$f_5(x) = max((w_i)^T x + b_i)$$

优点：

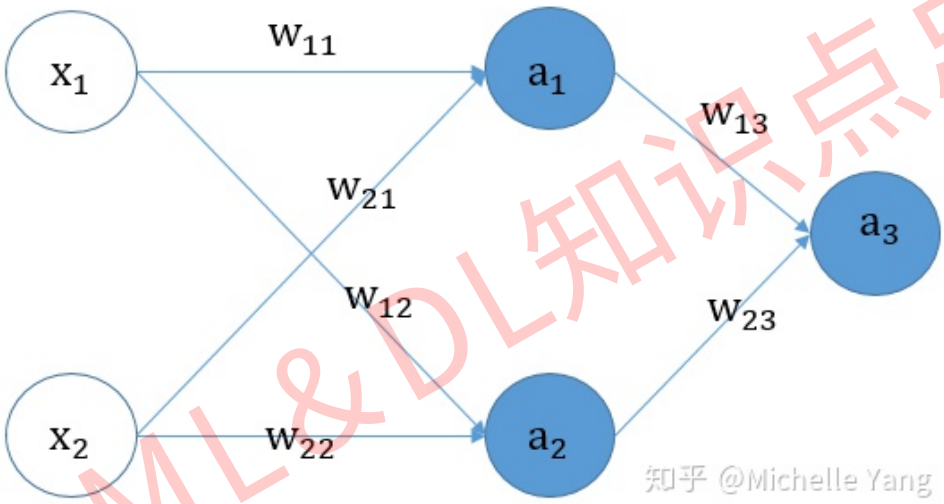
- maxout的拟合能力是非常强的，它可以拟合任意的的凸函数
- Maxout能够缓解梯度消失
- 规避了ReLU神经元死亡的缺点

缺点：

- 增加了参数和计算量

2．神经网络训练时是否可以将所有的参数初始化为0

为了说明这个问题，我们以一个简单的神经网络为例，该神经网络只有1层隐藏层，包含2个神经元，其具体的神经网络的结构图如下图所示：



神经网络的前向传播

$$a_1 = g(w_{11}x_1 + w_{21}x_2 + b_1)$$
$$a_2 = g(w_{12}x_1 + w_{22}x_2 + b_2)$$
$$a_3 = g(w_{13}a_1 + w_{23}a_2 + b_3)$$
$$L = -ylog(a_3) - (1-y)\log(1-a_3)$$

神经网络的反向传播所能用到的导数公式：

$$\frac{\partial L}{\partial a_3} = -\frac{y}{a_3} + \frac{1-y}{1-a_3}$$
$$\frac{\partial L}{\partial w_{13}} = (a_3 - y) * a_1$$
$$\frac{\partial L}{\partial w_{23}} = (a_3 - y) * a_2$$
$$\frac{\partial L}{\partial b_3} = (a_3 - y)$$
$$\frac{\partial L}{\partial a_1} = (a_3 - y) * w_{13}$$
$$\frac{\partial L}{\partial a_2} = (a_3 - y) * w_{23}$$
$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial a_2} * a_2' * x_1$$
$$\frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial a_2} * a_2' * x_2$$
$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial a_2} * a_2'$$
$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial a_1} * a_1' * x_1$$
$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial a_1} * a_1' * x_2$$
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_1} * a_1'$$

模型所有权重w初始化为0，所有偏置b初始化为0：

在此情况下，前向传播计算过程中，$a_1 = g(0), a_2 = g(0), a_3 = sigmoid(0)$，在反向传播进行参数更新的时候，会发现由于 a_1 和 a_2 均相等，会导致 dw_{13} 和 dw_{23} 相等，更新时 w_{13} 和 w_{23} 相等，出现权重的对称性。

由于所有权重初始化为0，这样在第一个batch反向传播的时候会导致 da_1 和 da_2 相等且为0，这样导致的结果是什么呢？第一个batch反向传播除了 w_{13} 和 w_{23} 以及 能够得到更新，其余权重均得不到更新。

当第二个batch传给神经网络之后，由于 w_{13} 和 w_{23} 相等但不为0，会导致 da_1 和 da_2 相等，这样会导致 w_{11} 和 w_{12} 相等， w_{21} 和 w_{22} 相等。

依次类推，无论训练多少次，无论我们隐藏层神经元有多少个，由于权重的对称性，我们的隐层的神经单元输出始终不变，出现隐藏神经元的对称性。我们希望不同神经元能够有不同的输出，这样的神经网络才有意义。

简单来说，就会出现**同一隐藏层所有神经元的输出都一致，对于后期不同的batch，每一隐藏层的权重都能得到更新，但是存在每一隐藏层的隐藏神经元权重都是一致的，多个隐藏神经元的作用就如同1个神经元。**

模型所有权重w初始化为0，所有偏置b随机初始化：

在此情况下，第一个batch前向传播：$a_1 = g(_1), a_2 = g(_2), a_3 = sigmoid(_3)$，在反向传播的过程中，由于 da_1 与 da_2 均为0，导致 $dw_{11}\ dw_{12}\ dw_{21}\ dw_{22}$ 均为0，则 $w_{11}\ w_{12}\ w_{21}\ w_{22}$ 得不到更新，为0，模型参数能够得到更新的只有 $w_{13}\ w_{23}$ 和 b_3 .

同理，在第二个batch在反向传播的过程中，由于 w_{13} 和 w_{23} 不为0，导致所有的参数都能够得到更新。这种方式存在更新较慢、梯度消失、梯度爆炸等问题，在实践中，通常不会选择此方式。

模型所有的权重w随机初始化，所有偏置b初始化为0：
在反向传播的过程中所有权重的导数都不相同，所以权重和偏置b都能得到更新。

3．梯度消失、梯度爆炸的原因及解决方法

- **梯度消失的原因：**
  a．网络层数过于深：反向传播不断更新参数的过程是一种连乘机制，随着网络层数的加深，连乘的值越来越多，当很多小于1的值相乘时，会导致乘积的最终结果（梯度）非常小，最终导致输入侧的梯度更新慢或者无法更新。
  b．使用了不恰当的激活函数：如sigmoid函数，该函数可将任意值映射到（0, 1）范围之内，求导的结果是f’(x)=f(x)(1−f(x))，导数在（0, 0.25）之间。在反向传播过程中，一定会对外层sigmoid函数求导，很多接近0的数值相乘会导致梯度消失。
- **梯度爆炸原因：**
  a．网络层数过于深：反向传播不断更新参数的过程是一种连乘机制，随着网络层数的

加深，连乘的值越来越多，当很多大于1的值相乘时，会导致乘积的最终结果（梯度）非常大，最终导致输入侧的梯度过于大，以至于溢出，导致 NaN 值出现。

b．权重初始化值过于大：将w初始化为一个较大的值时，例如>10的值，那么从输出层到输入层每一层都会有一个s‘(zn)*wn的增倍，当s‘(zn)为0.25时s‘(zn)*wn>2.5，同梯度消失类似，当神经网络很深时，梯度呈指数级增长，最后到输入侧时，梯度将会非常大，最终会得到一个非常大的权重更新值。

- **解决方法：**

a．l1、l2正则化：如果发生梯度爆炸，那么权值就会变的非常大，通过正则化项来约束权重的大小，可以在一定程度上降低梯度爆炸的发生。

b．梯度剪切：梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。

c．选择合适的激活函数：如：relu函数的导数在正数部分是恒等于1的，因此在深层网络中使用relu激活函数就不会导致梯度消失和爆炸的问题。

d．batch normalization：通过对每一层的输出规范为均值和方差一致的，消除了权重参数放大缩小带来的影响，进而解决梯度消失和爆炸的问题，或者可以理解为BN将输出从饱和区拉倒了非饱和区。

e．残差网络：相比较于以前直来直去的网络结构，残差中有很多跨层连接结构，这样的结构在反向传播中具有很大的好处，可以避免梯度消失。

f．LSTM的“门（gate）”结构：LSTM的结构设计可以改善RNN中的梯度消失的问题。主要原因在于LSTM内部的门结构，通过改善一条路径上（Cell State）的梯度问题拯救了总体的远距离梯度。

g．pre-training+fine-tunning：基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练”（pre-training）；在预训练完成后，再对整个网络进行“微调”（fine-tunning）。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优。

4．过拟合、欠拟合

- **欠拟合出现原因**

a．模型复杂度过低

b．特征量过少

- **欠拟合的情况比较容易克服，常见解决方法有**

a．增加新特征，可以考虑加入进特征组合、高次特征，来增大假设空间

b．添加多项式特征，这个在机器学习算法里面用的很普遍，例如将线性模型通过添加二次项或者三次项使模型泛化能力更强

c．减少正则化参数，正则化的目的是用来防止过拟合的，但是模型出现了欠拟合，则需要减少正则化参数

d．使用非线性模型，比如核SVM 、决策树、深度学习等模型

e．调整模型的容量(capacity)，通俗地，模型的容量是指其拟合各种函数的能力

f．容量低的模型可能很难拟合训练集；使用集成学习方法，如Bagging ,将多个弱学习器Bagging

- **过拟合出现原因**

a．建模样本选取有误，如样本数量太少，选样方法错误，样本标签错误等，导致选取的样本数据不足以代表预定的分类规则

b．样本噪音干扰过大，使得机器将部分噪音认为是特征从而扰乱了预设的分类规则

c．假设的模型无法合理存在，或者说是假设成立的条件实际并不成立

d．参数太多，模型复杂度过高

e．对于决策树模型，如果我们对于其生长没有合理的限制，其自由生长有可能使节点只包含单纯的事件数据(event)或非事件数据(no event)，使其虽然可以完美匹配（拟合）训练数据，但是无法适应其他数据集

f．对于神经网络模型：a)对样本数据可能存在分类决策面不唯一，随着学习的进行，BP算法使权值可能收敛过于复杂的决策面；b)权值学习迭代次数足够多(Overtraining)，拟合了训练数据中的噪声和训练样例中没有代表性的特征

- **过拟合的解决方案**

a．正则化（Regularization）（L1和L2）

b．数据扩增，即增加训练数据样本

c．Dropout

d．Early stopping

e．使用简单的模型

f．减少特征数量

5．交叉验证

- 我们可以把整个数据集分成两部分，一部分用于训练，一部分用于验证，这也就是我们经常提到的训练集（training set）和测试集（test set）。

缺点:

最终模型与参数的选取将极大程度依赖于你对训练集和测试集的划分方法

该方法只用了部分数据进行模型的训练

- 我们现在只用一个数据作为测试集，其他的数据都作为训练集，并将此步骤重复N次（N为数据集的数据数量）。

缺点:

计算量过于大

- 我们每次的测试集将不再只包含一个数据，而是多个，具体数目将根据K的选取决定。比如，如果K=5

特点:

K越大，每次投入的训练集的数据越多，模型的Bias越小。但是K越大，又意味着每一次选取的训练集之前的相关性越大（考虑最极端的例子，当k=N，也就是在LOOCV里，每

次都训练数据几乎是一样的）。而这种大相关性会导致最终的test error具有更大的Variance。

6．损失函数(Loss Function)和成本函数(Cost Function)之间有什么区别

损失函数用于单个训练样本。它有时也称为误差函数(error function)。另一方面，成本函数是整个训练数据集的平均损失(average function)。优化策略旨在最小化成本函数。

7．分类损失函数与回归损失函数

- **回归损失：**
  - MSE:

每个训练样本的平方误差损失(也称为**L2 Loss**)是实际值和预测值之差的平方：

$$L = \left(y - f(x)\right)^2$$

相应的成本函数是这些**平方误差的平均值**(MSE)。

二次函数仅具有全局最小值。由于没有局部最小值，所以我们永远不会陷入它。因此，可以保证梯度下降将收敛到全局最小值(如果它完全收敛)。
MSE损失函数通过平方误差来惩罚模型犯的大错误。把一个比较大的数平方会使它变得更大。但有一点需要注意，这个属性使MSE成本函数对异常值的健壮性降低。因此，如果我们的数据容易出现许多的异常值，则不应使用这个它。

  - MAE：

每个训练样本的绝对误差是预测值和实际值之间的距离，与符号无关。绝对误差也称为**L1 Loss**：

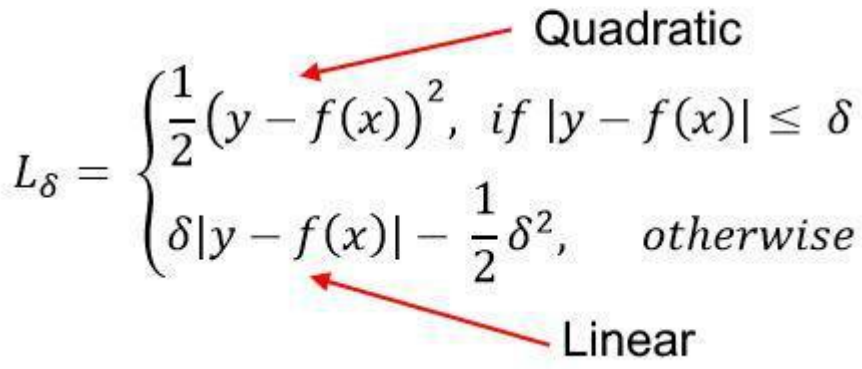
$$L = |y - f(x)|$$

正如我之前提到的，成本是这些绝对误差的平均值(MAE)。

与MSE相比，MAE成本对异常值更加健壮。但是，在数学方程中处理绝对或模数运算符并不容易。我们可以认为这是MAE的缺点。

  - Huber损失

Huber损失结合了MSE和MAE的最佳特性。对于较小的误差，它是二次的，否则是线性的(对于其梯度也是如此)。Huber损失需要确定 δ 参数：



Huber损失对于异常值比MSE更强。它用于稳健回归(robust regression)，M估计法(M-estimator)和可加模型(additive model)。Huber损失的变体也可以用于分类。

- **二分类损失：**
  - 二元交叉熵损失

元素属于第1类(或正类)的概率=p元素属于第0类(或负类)的概率=1-p然后，输出标签y(可以取值0和1)的交叉熵损失和和预测概率p定义为：

$$L = -y * \log(p) - (1 - y) * \log(1 - p) = \begin{cases} -\log(1 - p), & if\ y = 0 \\ -\log(p), & if\ y = 1 \end{cases}$$

这也称为Log-Loss(对数损失)。为了计算概率p，我们可以使用sigmoid函数。这里，z

  - hinge损失

Hinge损失主要用于带有类标签-1和1的支持向量机(SVM)。因此，请确保将数据集中"恶性"类的标签从0更改为-1。

> Hinge损失不仅会惩罚错误的预测，还会惩罚不自信的正确预测。

数据对(x，y)的Hinge损失如图：

$$L = \max(0, 1 - y * f(x))$$

Hinge损失简化了SVM的数学运算，同时最大化了损失(与对数损失(Log-Loss)相比)。当我们想要做实时决策而不是高度关注准确性时，就可以使用它。

- **多分类损失**
  - 多分类交叉熵损失：

多分类交叉熵损失是二元交叉熵损失的推广。输入向量 X_i 和相应的one-hot编码目标向量 Y_i 的损失是：

$$L(X_i, Y_i) = -\sum_{j=1}^{c} y_{ij} * \log(p_{ij})$$

$$where\ Y_i\ is\ one-hot\ encoded\ target\ vector\ (y_{i1}, y_{i2}, \ldots, y_{ic}),$$

$$y_{ij} = \begin{cases} 1, & if\ i_{th}\ element\ is\ in\ class\ j \\ 0, & otherwise \end{cases}$$

$$p_{ij} = f(X_i) = Probability\ that\ i_{th}\ element\ is\ in\ class\ j$$

  - KL散度：

KL散度概率分布与另一个概率分布区别的度量。KL散度为零表示分布相同。

$$For\ probability\ distributions\ P\ and\ Q,$$

$$KL - Divergence\ of\ P\ from\ Q\ is\ given\ by$$

$$D_{KL}(P||Q) = \begin{cases} -\sum_x P(x).log\frac{Q(x)}{P(x)} = \sum_x P(x).log\frac{P(x)}{Q(x)}, & for\ discrete\ distributions \\ -\int P(x).log\frac{Q(x)}{P(x)}.dx = \int P(x).log\frac{P(x)}{Q(x)}.dx, & for\ continuous\ distributions \end{cases}$$

$$= Expectation\ of\ logaritmic\ difference\ between\ P\ and\ Q\ with\ respect\ to\ P$$

$$D_{KL}(P||Q) = -\sum_x \big(P(x).logQ(x) - P(x).logP(x)\big) = H(P,Q) - H(P,P)$$

$$H(P,P)\ is\ the\ entropy\ of\ P$$

$$H(P,Q)\ is\ the\ cross-entropy\ of\ P\ and\ Q$$

与多分类分类相比，KL散度更常用于逼近复杂函数。我们在使用变分自动编码器(VAE)等深度生成模型时经常使用KL散度。

8．判别模型与生成模型的区别

判别式模型评估对象是最大化条件概率p(y|x)并直接对其建模，判别式模型直接通过解在满足训练样本分布下的最优化问题得到模型参数，主要用到拉格朗日乘算法、梯度下降法，常见的判别式模型如最大熵模型、CRF、LR、SVM、KNN、DT、boosting方法等；而生成式模型先经过贝叶斯转换成Y = argmax p(y|x) = argmax p(x|y)*p(y)，然后分别学习p(y)和p(x|y)的概率分布，主要通过极大似然估计的方法学习参数，如NGram、HMM、Naive Bayes。

9．类别不平衡如何处理

- **欠采样：**

我之前以为欠采样就是直接对训练集中多数类样本进行“欠采样”（undersampling），即去除一些多数类中的样本使得正例、反例数目接近，然后再进行学习。但这也存在一些问题，由于采样的样本集合要少于原来的样本集合，因此会造成一些信息缺失，即**将多数类样本删除有可能会导致分类器丢失有关多数类的重要信息。**后来了解后发现这只是最简单的随机欠采样。针对其方法的改进还有很多，其中具有代表性的是结合集

成学习思想的EasyEnsemble和结合boosting的BalanceCascade。

1、EasyEnsemble，利用模型融合的方法（Ensemble）：

多次过采样（放回采样，这样产生的训练集才相互独立）产生多个不同的训练集，进而训练多个不同的分类器，通过组合多个分类器的结果得到最终的结果。简单的最佳实践是建立n个模型，每个模型使用少数类的所有样本和多数类的n个不同样本。假设二分类数据集的正负类比例为50000:1000，最后要得到10个模型，那么将保留负类的1000个样本，并随机采样得到10000个正类样本。

然后，将10000个样本成10份，每一份与负类样本组合得到新的子训练集，训练10个不同的模型。

2、BalanceCascade，利用增量训练的思想（Boosting）：

先通过一次下采样产生训练集，训练一个分类器，对于那些分类正确的多数类样本不放回，然后对这个更小的多数类样本下采样产生训练集，训练第二个分类器，以此类推，最终组合所有分类器的结果得到最终结果。

- **过采样：**

过采样就是对训练集里的少数类进行“过采样”（oversampling），即增加一些少数类样本使得正、反例数目接近，然后再进行学习。最简单的当然还是随机过采样，即随机的复制少数类样本使数据集趋于均衡。但这造成模型训练复杂度加大。另一方面也容易造成模型的过拟合问题，因为随机过采样是简单的对初始样本进行复制采样，这就使得**学习器学得的规则过于具体化，不利于学习器的泛化性能，造成过拟合问题。**所以针对此问题的一个代表性改进工作是SMOTE。

SMOTE全称是Synthetic Minority Oversampling即合成少数类过采样技术。SMOTE算法是对随机过采样方法的一个改进算法，由于随机过采样方法是直接对少数类进行重采用，会使训练集中有很多重复的样本，容易造成产生的模型过拟合问题。而SOMT算法的基本思想是对每个少数类样本 x_i ，从它的最近邻中随机选择一个样本 $\hat{x_i}$ （ $\hat{x_i}$ 是少数类中的一个样本），然后在 x_i 和 $\hat{x_i}$ 之间的连线上随机选择一点作为新合成的少数类样本。

SMOTE算法合成新少数类样本的算法描述如下：

1).对于少数类中的每一个样本 x_i ，以欧氏距离为标准计算它到少数类样本集 S_{min} 中所有样本的距离，得到其k近邻。

2).根据样本不平衡比例设置一个采样比例以确定采样倍率N，对于每一个少数类样本 x_i ，从其k近邻中随机选择若干个样本，假设选择的是 $\hat{x_i}$ 。

3).对于每一个随机选出来的近邻 $\hat{x_i}$ ，分别与 x_i 按照如下公式构建新的样本。

$$x_{new} = \text{ɕ}_i + rand(0,1) \times (\hat{x_i} - x_i)$$

- **调整样本权重：**

调整样本权重和修改分类阈值在[1]中统一称为**代价敏感学习**，我理解的就是在学习中为少数类样本赋予更高的权重，比如在神经网络中，少数类产生的误差损失对网络权重更新力度更大。可以达到和随机过采样相同的效果，且不会使训练复杂度增大。

- **修改分类阈值：**

我理解的修改分类阈值就是，因为数据集存在类别不均衡的情况，所以直接训练分类器会使得模型在预测时更偏向于多数类，所以不再以0.5为分类阈值，而是针对少数类在模型仅有较小把握时就将样本归为少数类。

- **损失函数的选择：**

Focal loss是一个能较好适应类别不均衡的损失函数，对于二分类问题Focal loss计算如下：

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t).$$

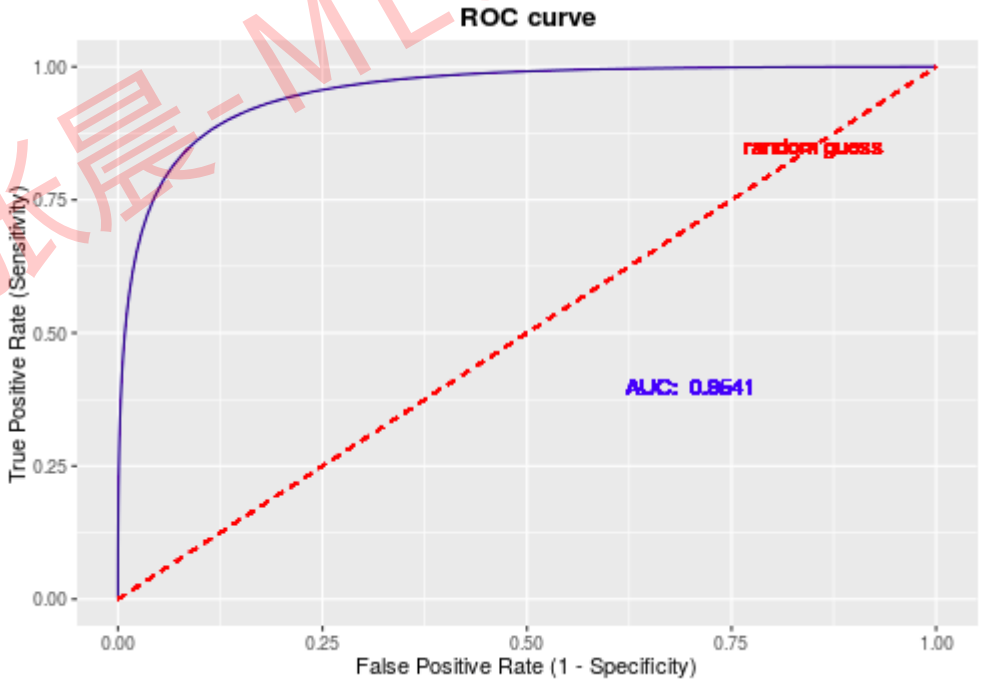
对于那些概率较大的样本 $(1-p_t)^{\gamma}$ 趋近于0，可以降低它的loss值，而对于真实概率比较低的困难样本，$(1-p_t)^{\gamma}$ 对他们的loss影响并不大，这样一来我们可以通过降低简单样本loss的方法提高困难样本对梯度的贡献。同时为了提高误分类样本的权重，最终作者为Focal loss增加权重，Focal loss最终长这样：

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t).$$

当然Focal loss对多分类的任务也同样适用。

10．类别不平衡的评价指标

ROC曲线：



ROC曲线常用于二分类问题中的模型比较，主要表现为一种真正例率 (TPR) 和假正例率 (FPR) 的权衡。具体方法是在不同的分类阈值 (threshold) 设定下分别以TPR和FPR为纵、横轴作图。由ROC曲线的两个指标，$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$，$FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$ 可以看出，当一个样本被分类器判为正例，若其本身是正例，则TPR增加；若其本身是负例，则FPR增加，因此ROC曲线可以看作是随着阈值的不断移动，所有样本中正例与负例之间的“对抗”。曲线越靠近左上角，意味着越多的正例优先于负例，模型的整体表现也就越好。

ROC曲线的绘制方法：假设有P个正例，N个反例，首先拿到分类器对于每个样本预测为正例的概率，根据概率对所有样本进行逆序排列，然后将分类阈值设为最大，即把所有样本均预测为反例，此时图上的点为 (0,0)。然后将分类阈值依次设为每个样本的预测概率，即依次将每个样本划分为正例，如果该样本为真正例，则TP+1，即$TPR + \frac{1}{P}$；如果该样本为负例，则FP+1，即$FPR + \frac{1}{N}$。最后的到所有样本点的TPR和FPR值，用线段相连。

1. 兼顾正例和负例的权衡。因为TPR聚焦于正例，FPR聚焦于与负例，使其成为一个比较均衡的评估方法。
2. ROC曲线选用的两个指标，$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$，$FPR = \frac{FP}{N} = \frac{FP}{FP+TN}$，都不依赖于具体的类别分布。

   注意TPR用到的TP和FN同属P列，FPR用到的FP和TN同属N列，所以即使P或N的整体数量发生了改变，也不会影响到另一列。也就是说，即使正例与负例的比例发生了很大变化，ROC曲线也不会产生大的变化，而像Precision使用的TP和FP就分属两列，则易受类别分布改变的影响。

ROC曲线的缺点

1. 上文提到ROC曲线的优点是不会随着类别分布的改变而改变，但这在某种程度上也是其缺点。因为负例N增加了很多，而曲线却没变，这等于产生了大量FP。像信息检索中如果主要关心正例的预测准确性的话，这就不可接受了。
2. 在类别不平衡的背景下，负例的数目众多致使FPR的增长不明显，导致ROC曲线呈现一个过分乐观的效果估计。ROC曲线的横轴采用FPR，根据FPR = $\frac{FP}{N} = \frac{FP}{FP+TN}$，当负例N的数量远超正例P时，FP的大幅增长只能换来FPR的微小改变。结果是虽然大量负例被错判成正例，在ROC曲线上却无法直观地看出来。（当然也可以只分析ROC曲线左边一小段）
   举个例子，假设一个数据集有正例20，负例10000，开始时有20个负例被错判，$FPR = \frac{20}{20+9980} = 0.002$，接着又有20个负例错判，$FPR_2 = \frac{40}{40+9960} = 0.004$，在ROC曲线上这个变化是很细微的。而与此同时Precision则从原来的0.5下降到了0.33，在PR曲线上将会是一个大幅下降。

AUC值：

ROC曲线围成的面积 (**即AUC**)可以解读为：从所有正例中随机选取一个样本A，再从所有负例中随机选取一个样本B，分类器将A判为正例的概率比将B判为正例的概率大的可能性。可以看到位于随机线上方的点(如图中的A点)被认为好于随机猜测。在这样的点上TPR总大于FPR，意为正例被判为正例的概率大于负例被判为正例的概率。

从另一个角度看，由于画ROC曲线时都是先将所有样本按分类器的预测概率排序，所以AUC反映的是分类器对样本的排序能力，依照上面的例子就是A排在B前面的概率。AUC越大，自然排序能力越好，即分类器将越多的正例排在负例之前。

PR曲线：

PR曲线展示的是Precision vs Recall的曲线，PR曲线与ROC曲线的相同点是都采用了TPR (Recall)，都可以用AUC来衡量分类器的效果。不同点是ROC曲线使用了FPR，而PR曲线使用了Precision，因此PR曲线的两个指标都聚焦于正例。类别不平衡问题中由于主要关心正例，所以在此情况下PR曲线被广泛认为优于ROC曲线。

PR曲线的绘制与ROC曲线类似，PR曲线的AUC面积计算公式为：

$$\sum_n (R_n - R_{n-1}) P_n$$

1. ROC曲线由于兼顾正例与负例，所以适用于评估分类器的整体性能，相比而言PR曲线完全聚焦于正例。
2. 如果有多份数据且存在**不同**的类别分布，比如信用卡欺诈问题中每个月正例和负例的比例可能都不相同，这时候如果只想单纯地比较分类器的性能且剔除类别分布改变的影响，则ROC曲线比较适合，因为类别分布改变可能使得PR曲线发生变化时好时坏，这种时候难以进行模型比较；反之，如果想测试不同类别分布下对分类器的性能的影响，则PR曲线比较适合。
3. 如果想要评估在**相同**的类别分布下正例的预测情况，则宜选PR曲线。
4. 类别不平衡问题中，ROC曲线通常会给出一个乐观的效果估计，所以大部分时候还是PR曲线更好。
5. 最后可以根据具体的应用，在曲线上找到最优的点，得到相对应的precision，recall，f1 score等指标，去调整模型的阈值，从而得到一个符合具体应用的模型。

11．类别性特征处理

- 独热编码
- 序号编码
- 计数编码：比如类别A在训练集中出现了100次则编码为100
- 树模型自动处理：lightgbm和CatBoost，可以直接处理categorical feature
- 目标编码：例如类别A对应的标签1有100个，标签2有100个，标签3有100个，则可以编码为【1/3,1/3,1/3】
- 二进制编码：转化成序号编码，再转为二进制形式

12．缺失值的处理

删除缺少值的行：

可以通过删除具有空值的行或列来处理缺少的值。如果列中有超过一半的行为null，则可以删除整个列。也可以删除具有一个或多个列值为null的行。

优点：

- 可以创建一个健壮的模型。

缺点：

- 大量信息丢失。
- 如果与完整的数据集相比，缺失值的百分比过大，则效果不佳。

用平均值/中位数估算缺失值：

数据集中具有连续数值的列可以替换为列中剩余值的平均值、中值或众数。与以前的方法相比，这种方法可以防止数据丢失。替换上述两个近似值（平均值、中值）是一种处理缺失值的统计方法。

优点：

- 防止导致删除行或列的数据丢失
- 在一个小的数据集上运行良好，并且易于实现。

缺点：

- 仅适用于数值连续变量。
- 不考虑特征之间的协方差。

分类列的插补方法：

如果缺少的值来自分类列（字符串或数值），则可以用最常见的类别替换丢失的值。如果缺失值的数量非常大，则可以用新的类别替换它，比如性别，有男、女、缺失三种情况，则映

射成3个变量：是否男、是否女、是否缺失。

优点：

- 防止导致删除行或列的数据丢失
- 在一个小的数据集上运行良好，并且易于实现。
- 通过添加唯一类别来消除数据丢失

缺点：

- 仅适用于分类变量。
- 在编码时向模型中添加新特征，这可能会导致性能较差

算法处理：

这种情况需要考虑算法的适用性，比如线性回归、支撑向量回归等需要进行缺失值处理，否则算法无法处理；而决策树等以树为基准的算法则不需要进行缺失值处理；具体需要针对不同算法来确定。

优点：

- 不需要处理每列中缺少的值，因为ML算法可以有效地处理它

缺点：

- scikit learn库中没有这些ML算法的实现。

缺失值预测：

用其他变量做预测模型来算出缺失变量

优点：

- 给出了比以前的方法更好的结果
- 考虑缺失值列与其他列之间的协方差。

缺点：

- 用其他变量做预测模型来算出缺失变量

13．生成模型的不可控

14．NLP如何做数据增强

- 词汇替换篇
  - 基于词典的替换方法：基于同义词替换的方法是从句子中以一定的概率随机选取一个单词，利用一些同义词数据库（注：英文可以用 WordNet 数据库，中文可以用 synonyms python 同义词词典） 将其替换成对应的同义词。
  - 基于词向量的替换方法：通过利用预先训练好的词向量（eg:Word2Vec、GloVe、FastText等），使用嵌入空间中最近的相邻单词替换句子中的某些单词。
  - 基于 MLM 的替换方法：像BERT、ROBERTA和ALBERT这样的Transformer模型已经接受了大量的文本训练，使用一种称为“Masked Language Modeling”的预训练，即模型必须根据上下文来预测遮盖的词汇。这可以用来扩充一些文本。例如，我们可以使用一个预训练的BERT模型并屏蔽文本的某些部分。然后，我们使用BERT模型来预测遮蔽掉的token。使用mask预测来生成文本的变体。与之前的方法相比，生成的文本在语法上更加连贯，因为模型在进行预测时考虑了上下文。

- 词汇插入篇
  - 随机插入法：通过在 query 里面随机插入一个或多个新词汇、相应的拼写错误、符号等噪声的方式提高 训练模型的健壮性。
- 词汇交换篇
  - 随机交换法：通过在 query 里面随机交换一个或多个词汇的方式提高 训练模型的健壮性。
- 词汇删除篇
  - 随机删除法：通过在 query 里面随机删除一个或多个词汇的方式提高 训练模型的健壮性。
- 回译篇
  - 回译法：利用百度翻译、谷歌翻译等在线翻译器来解释文本，并重新训练文本
- 交叉增强篇
  - 交叉增强：将 tweets 切分未为两部分，两个具有相同极性的随机推文进行交换。这个方法的假设是，即使结果是不符合语法和语义的，新文本仍将保留情感的极性。
- 语法树篇
  - 语法树：解析和生成原始句子的依赖关系树，使用规则对其进行转换，并生成改写后的句子。举例说明：在不改变句子的含义的情况下将句子从主动转化为被动语态的方式。

15．L1正则化与L2正则化的区别

结构风险最小化： 在经验风险最小化的基础上（也就是训练误差最小化），尽可能采用简单的模型，以此提高泛化预测精度。

优化目标：

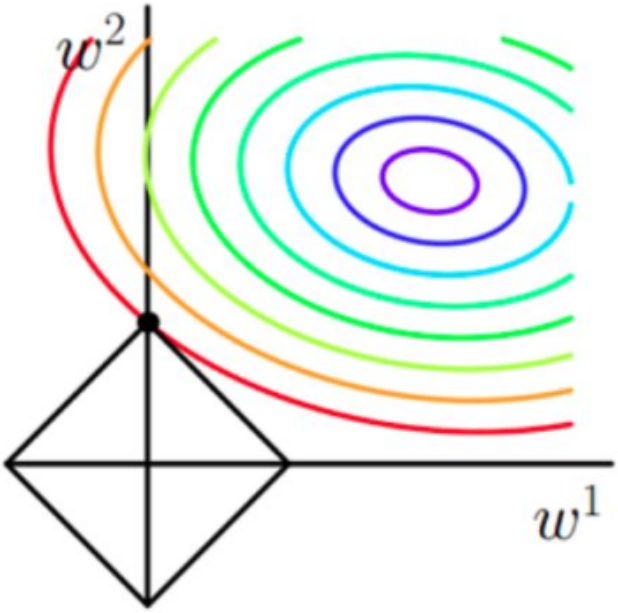
$$\min\ 1/N * \sum_{i=1}^{N}(y_i - \omega^T x_i)^2 \quad \text{式子（1）}$$

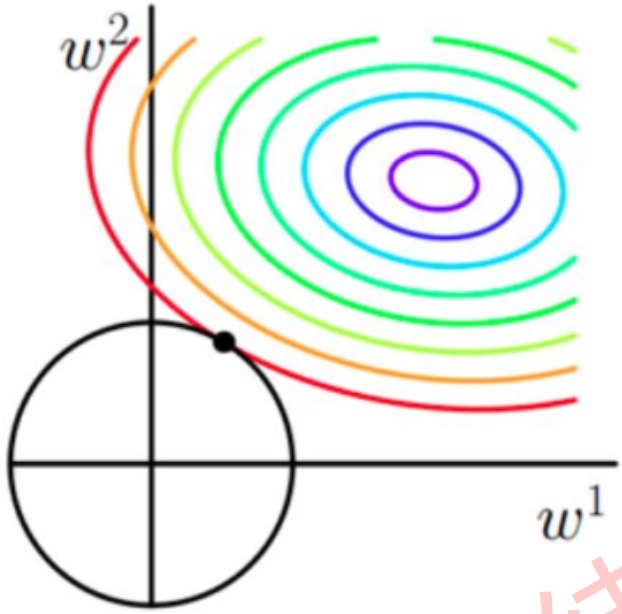
加上L1正则项（lasso回归）：

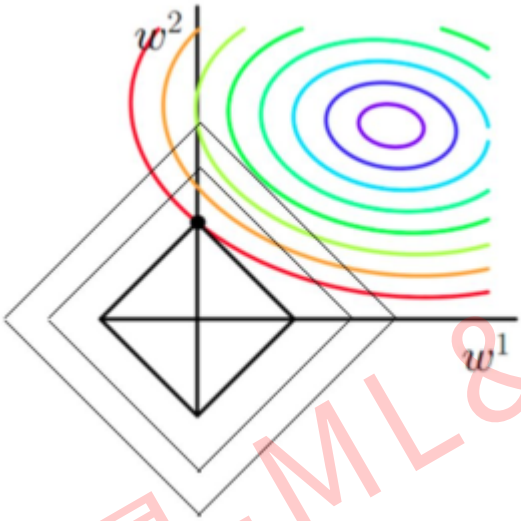
$$\min\ 1/N * \sum_{i=1}^{N}(y_i - \omega^T x_i)^2 + C||\omega||_1 \quad \text{式子（2）}$$

加上L2正则项（岭回归）：

$$\min\ 1/N * \sum_{i=1}^{N}(y_i - \omega^T x_i)^2 + C||\omega||_2^2 \quad \text{式子（3）}$$







- 如果不加L1和L2正则化的时候，对于线性回归这种目标函数凸函数的话，我们最终的结果就是最里边的紫色的小圈圈等高线上的点。
- 当加入L1正则化的时候，我们先画出 $|\omega_1| + |\omega_2| = F$ 的图像，也就是一个菱形，代表这些曲线上的点算出来的 1范数 $|\omega_1| + |\omega_2|$ 都为F。那我们现在的目标是不仅是原曲线算得值要小（越来越接近中心的紫色圈圈），还要使得这个菱形越小越好（F越小越好）。那么还和原来一样的话，过中心紫色圈圈的那个菱形明显很大，因此我们要取到一个恰好的值。那么如何求值呢?

我们可以看出，最终加入L1范数得到的解，一定是某个菱形和某条原函数等高线的切点。现在有个比较重要的结论来了，我们经过观察可以看到，几乎对于很多原函数等高曲线，和某个菱形相交的时候及其容易相交在坐标轴（比如上图），也就是说最终的结果，解的某些维度及其容易是0，比如上图最终解是 ，这也就是我们所说的L1更容易得到稀疏解（解向量中0比较多）的原因。

当加入L2正则化的时候，分析和L1正则化是类似的，也就是说我们仅仅是从菱形变成了圆形而已，同样还是求原曲线和圆形的切点作为最终解。当然与L1范数比，我们这样求的L2范数的从图上来看，不容易交在坐标轴上，但是仍然比较靠近坐标轴。因此这也就是我们老说的，L2范数能让解比较小（靠近0），但是比较平滑（不等于0）。

16．为什么参数越小代表模型越简单?

越是复杂的模型，越是尝试对所有样本进行拟合，包括异常点。这就会造成在较小的区间中产生较大的波动，这个较大的波动也会反映在这个区间的导数比较大。只有越大的参数才可能产生较大的导数。因此参数越小，模型就越简单。

17．实现参数的稀疏有什么好处?

因为参数的稀疏，在一定程度上实现了特征的选择。一般而言，大部分特征对模型是没有贡献的。这些没有用的特征虽然可以减少训练集上的误差，但是对测试集的样本，反而会产生干扰。稀疏参数的引入，可以将那些无用的特征的权重置为0。

18．L1范数和L2范数为什么可以避免过拟合?

L1范数:

L1范数符合拉普拉斯分布，是不完全可微的，表现在图像上会有很多角出现，这些角和目标函数的接触机会远大于其他部分。就会造成最优值出现在坐标轴上，因此就会导致某一维的权重为0 ，产生稀疏权重矩阵，进而防止过拟合。

L2范数:

L2范数符合高斯分布，是完全可微的。和L1相比，图像上的棱角被圆滑了很多。一般最优值不会在坐标轴上出现。在最小化正则项时，可以是参数不断趋向于0.最后获得很小的参数。