机器学习与深度学习面试系列三（优化算法）

有监督学习涉及的损失函数有哪些?

对于分类问题，常用的损失函数包括：

1. 均方差损失函数(Mean Squared Error Loss)： $J_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$

2. 平均绝对误差损失(Mean Absolute Error Loss): $J_{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}|$

3. Huber Loss:

$$J_{huber} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}_{|y_i-\hat{y_i}|\leq\delta}\frac{(y_i-\hat{y_i})^2}{2} + \mathbb{I}_{|y_i-\hat{y_i}|>\delta}(\delta|y_i-\hat{y_i}| - \frac{1}{2}\delta^2)$$

对于二分类问题，最自然的损失函数是0-1损失函数，它能够直观地刻画分类的错误率，但是由于其非凸、非光滑的特点， 使得算法很难直接对该函数进行优化，所以通常使用它的代理函数：

1. Hinge损失函数： $J_{hinge} = \sum_{i=1}^{N} max\{0, 1 - sgn(y_i)\hat{y_i}\}$

2. 交叉熵 (Cross Entropy)损失函数: $J_{BCE} = -\sum_{i=1}^{N} y_i log(\hat{y_i}) + (1-y_i)log(1-\hat{y_i})$ ，拓展到多分类为： $J_{CE} = -\sum_{i=1}^{N} y_i^{c_i} log({\hat{y_i}}^{c_i})$ ，其中 c_i 是样本 x_i 的目标分类。用交叉熵损失函数后，在0-1损失达到0后还能持续下降很长一段时间，拉开不同类别的距离以改进分类器的鲁棒性。

这些代理函数都是0-1损失函数的光滑凸上界。

均方差损失函数和高斯先验的关系?

假设**模型预测与真实值之间的误差服从标准高斯分布**（ $\mu = 0, \sigma = 1$ ），则给定一个 x_i 模型输出真实值 y_i 的概率为

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y_i-\hat{y_i})^2}{2}\right)$$

进一步我们假设数据集中 N 个样本点之间相互独立，则给定所有 $\boldsymbol{x}$ 输出所有真实值 $\boldsymbol{y}$ 的概率，即似然 Likelihood，为所有 $\boldsymbol{p(y_i|x_i)}$ 的累乘

$$L(x,y)=\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y_i-\hat{y}_i)^2}{2}\right)$$

通常为了计算方便，我们通常最大化对数似然 Log-Likelihood

$$LL(x,y)=\log(L(x,y))=-\frac{N}{2}\log 2\pi-\frac{1}{2}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$$

去掉与 $\boldsymbol{\hat{y}_i}$ 无关的第一项，然后转化为最小化负对数似然 Negative Log-Likelihood

$$NLL(x,y)=\frac{1}{2}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$$

可以看到这个实际上就是均方差损失的形式。也就是说**在模型输出与真实值的误差服从高斯分布的假设下，最小化均方差损失函数与极大似然估计本质上是一致的**，因此在这个假设能被满足的场景中（比如回归），均方差损失是一个很好的损失函数选择；当这个假设没能被满足的场景中（比如分类），均方差损失不是一个好的选择。

平均绝对误差损失函数和拉普拉斯先验的关系？

假设**模型预测与真实值之间的误差服从拉普拉斯分布 Laplace distribution（ $\mu=0,b=1$ ）**，则给定一个 $\boldsymbol{x_i}$ 模型输出真实值 $\boldsymbol{y_i}$ 的概率为

$$p(y_i|x_i)=\frac{1}{2}\exp(-|y_i-\hat{y}_i|)$$

与上面推导 MSE 时类似，我们可以得到的负对数似然实际上就是 MAE 损失的形式

$$L(x,y)=\prod_{i=1}^{N}\frac{1}{2}\exp(-|y_i-\hat{y}_i|)$$

$$LL(x,y)=-\frac{N}{2}-\sum_{i=1}^{N}|y_i-\hat{y}_i|$$

$$NLL(x,y)=\sum_{i=1}^{N}|y_i-\hat{y}_i|$$

均方差损失函数 与 平均绝对误差损失函数 区别?

MSE 损失相比 MAE 通常可以更快地收敛，但 MAE 损失对于离群点更加健壮，即更加不易受到离群点影响。

当使用梯度下降算法时，MSE 损失的梯度为 $-\hat{y_i}$ ，而 MAE 损失的梯度为 ± 1 ，即 MSE 的梯度的 scale 会随误差大小变化，而 MAE 的梯度的 scale 则一直保持为 1，即便在绝对误差 $|y_i - \hat{y_i}|$ 很小的时候 MAE 的梯度 scale 也同样为 1，这实际上是非常不利于模型的训练的。

由于MAE 损失与绝对误差之间是线性关系，MSE 损失与误差是平方关系，当误差非常大的时候，MSE 损失会远远大于 MAE 损失。因此当数据中出现一个误差非常大的离群点时，MSE 会产生一个非常大的损失，对模型的训练会产生较大的影响。

Huber Loss 有什么特点?

Huber Loss 结合了 MSE 和 MAE 损失，在误差接近 0 时使用 MSE，使损失函数可导并且梯度更加稳定；在误差较大时使用 MAE 可以降低 outlier 的影响，使训练对 outlier 更加健壮。缺点是需要额外地设置一个 δ 超参数。

分类中为什么不用均方差损失?

交叉熵损失函数关于输入权重的梯度表达式与预测值与真实值的误差成正比且不含激活函数的梯度，而均方误差损失函数关于输入权重的梯度表达式中则含有，由于常用的sigmoid/tanh等激活函数存在梯度饱和区，使得MSE对权重的梯度会很小，参数w调整的慢，训练也慢，而交叉熵损失函数则不会出现此问题，其参数w会根据误差调整，训练更快，效果更好。

均方差损失的时候讲到实际上均方差损失假设了误差服从高斯分布，在分类任务下这个假设没办法被满足，因此效果会很差。实际上交叉熵损失函数可以看作是分类结果满足伯努利分布：

$$p(y_i = 1|x_i) = \hat{y_i}$$
$$p(y_i = 0|x_i) = 1 - \hat{y_i}$$

将两条式子合并成一条

$$p(y_i|x_i) = (\hat{y_i})^{y_i}(1 - \hat{y_i})^{1-y_i}$$

假设数据点之间独立同分布，则似然可以表示为

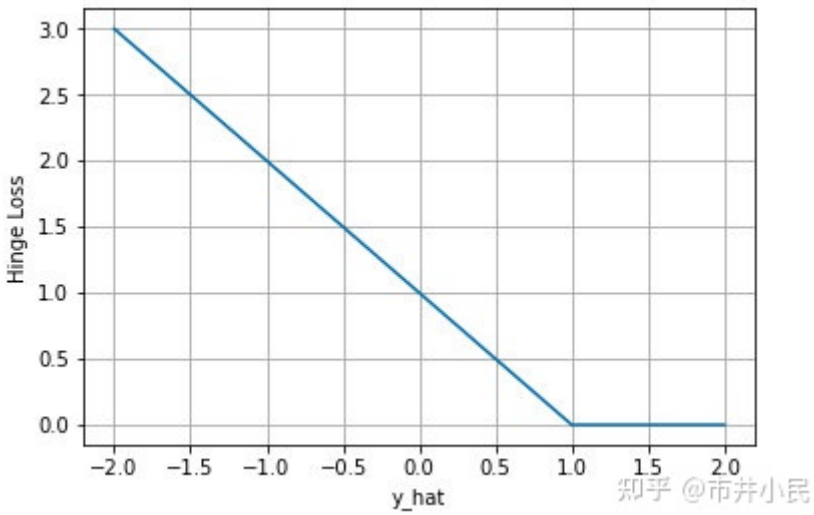
$$L(x,y)=\prod_{i=1}^{N}(\hat{y}_i)^{y_i}(1-\hat{y}_i)^{1-y_i}$$

对似然取对数，然后加负号变成最小化负对数似然，即为交叉熵损失函数的形式

$$NLL(x,y)=J_{CE}=-\sum_{i=1}^{N}(y_i\log(\hat{y}_i)+(1-y_i)\log(1-\hat{y}_i))$$

如何理解合页损失?

合页损失适用于 maximum-margin 的分类，支持向量机 Support Vector Machine (SVM) 模型的损失函数本质上就是 Hinge Loss + L2 正则化。



可以看到当 y 为正类时，模型输出负值会有较大的惩罚，当模型输出为正值且在 $(0,1)$ 区间时还会有一个较小的惩罚。即合页损失不仅惩罚预测错的，并且对于预测对了但是置信度不高的也会给一个惩罚，只有置信度高的才会有零损失。使用合页损失直觉上理解是要**找到一个决策边界，使得所有数据点被这个边界正确地、高置信地被分类**。[1]

无约束优化问题的优化方法有哪些?

经典的优化算法可以分为直接法和迭代法两大类。

直接法就是能够直接给出优化问题最优解的方法，它需要满足L(·)是凸函数，如果L(·)是凸函数，根据 $\nabla L(\theta^*)=0$ ，则： θ^* 是最优解。为了能够直接求解出 θ^* ，还需要满足第二个条件：上式有闭式解。例如线性回归的正规方程法。

迭代法又分为一阶法和二阶法。一阶法也称梯度下降法，二阶法也称为牛顿法。

什么是梯度?

梯度的方向是函数值增加最快的方向，梯度的相反方向是函数值减小的最快的方向。

为什么梯度的负方向是局部下降最快的方向?

对 $f(x+v)$ 进行一阶泰勒展开： $f(x+v) \approx f(x)+\nabla f(x)^T v$ ，移项得：
$f(x+v)-f(x) \approx \nabla f(x)^T v$ 。

$\nabla f(x)^T v$ 为函数值的变化量，我们要注意的是 $\nabla f(x)$ 和 v 均为向量， $\nabla f(x)^T v$ 也就是两个向量进行点积，而向量进行点积的最大值，也就是两者共线的时候，也就是说 v 的方向和 $\nabla f(x)$ 方向相同的时候，点积值最大。所以说明了梯度方向是函数局部上升最快的方向，也就证明了梯度的负方向是局部下降最快的方向!

牛顿法和梯度法有什么区别?

梯度下降法： $\overrightarrow{\mathbf{x}}_{k+1}=\overrightarrow{\mathbf{x}}_k-\epsilon \overrightarrow{\mathbf{g}}$

牛顿法： $\overrightarrow{\mathbf{x}}_{k+1}=\overrightarrow{\mathbf{x}}_k-\mathbf{H}^{-1} \overrightarrow{\mathbf{g}}$

1. 梯度法对目标函数进行一阶泰勒展开，梯度就是目标函数的一阶信息；
2. 牛顿法对目标函数进行二阶泰勒展开，Hessian矩阵就是目标函数的二阶信息。
3. 牛顿法的收敛速度一般要远快于梯度法，但是在高维情况下Hessian矩阵求逆的计算复杂度很大，而且当目标函数非凸时，牛顿法有可能会收敛到鞍点。
4. 因为梯度法旨在朝下坡移动，而牛顿法目标是寻找梯度为0的点。
5. 位于一个极小值点附近时，牛顿法比梯度下降法能更快地到达极小值点。
   如果在一个鞍点附近，牛顿法效果很差，因为牛顿法会主动跳入鞍点。而梯度下降法此时效果较好（除非负梯度的方向刚好指向了鞍点）。
6. 梯度下降法中，每一次 $\overrightarrow{\mathbf{x}}$ 增加的方向一定是梯度相反的方向 $-\epsilon_k \nabla_k$ 。增加的幅度由 ϵ_k （学习率） 决定，若跨度过大容易引发震荡。
   而牛顿法中，每一次 $\overrightarrow{\mathbf{x}}$ 增加的方向是梯度增速最大的反方向 $-\mathbf{H}_k^{-1} \nabla k$ （它通常情况下与梯度不共线）。增加的幅度已经包含在 $\mathbf{H}_k^{-1}$ 中（也可以乘以学习率作为幅度的系数）。

为什么深度学习中不使用牛顿法?

1. 牛顿法是对函数在对应点进行二阶泰勒展开导出的方法，如果当前位置离最小值点很远，那么二阶泰勒展开不能近似原函数。也就是说此时我们用牛顿法计算的方向可能偏差比较大。解决这个问题有一种方法就是先用梯度下降法，到最小值附近时再用牛顿法来加快收敛
2. 深度学习中参数量过大，对于海森矩阵的计算非常耗时。对于大型神经网络来说，几乎不可

用。
3. 神经网络中广泛的使用了ReLU激活函数，ReLU是非解析函数，也就是非处处可微的，而牛顿法是基于泰勒展开导出的方法。（泰勒展开要求函数是处处可微的）
4. 深度学习中损失函数的landscape过于复杂，存在很多鞍点，牛顿法有可能会收敛到鞍点。

什么是拟牛顿法?

在牛顿法的迭代中，需要计算海森矩阵的逆矩阵 $\mathbf{H}^{-1}$ ，这一计算比较复杂。可以考虑用一个 n 阶矩阵 $\mathbf{G}_k = G\left(\overrightarrow{\mathbf{x}}\right)$ 来近似代替 。

如果选择 G_k 作为 $\mathbf{H}_k^{-1}$ *的近似时，* $\mathbf{G}_k$ 同样要满足两个条件：

- $\mathbf{G}_k$ *必须是正定的。*
- $\mathbf{G}_k$ *满足下面的拟牛顿条件：* $\mathbf{G}_{k+1}\overrightarrow{\mathbf{y}}_k = \vec{\delta}_k$
  因为 $\mathbf{G_0}$ 是给定的初始化条件，所以下标从 $k+1$ 开始。

按照拟牛顿条件选择 $\mathbf{G}_k$ *作为* $\mathbf{H}_k^{-1}$ 的近似或者选择 $\mathbf{B}_k$ *作为* $\mathbf{H}_k$ 的近似的算法称为拟牛顿法

按照拟牛顿条件，在每次迭代中可以选择更新矩阵 $\mathbf{G}_{k+1} = \mathbf{G}_k + \Delta\mathbf{G}_k$

当训练数据量特别大时，经典的梯度下降法存在什么问题，需要做如何改进?

经典的梯度下降法在每次对模型参数进行更新时，需要遍历所有的训练数 据。当M很大时，这需要很大的计算量，耗费很长的计算时间，在实际应用中基本不可行。

随机梯度下降法用单个训练数据即可对模型参数进行一次更新，大大加快了收敛速率。该方法也非常适用于数据源源不断到来的在线更新场景。

为了降低随机梯度的方差，从而使得迭代算法更加稳定，也为了充分利用高度优化的矩阵运算操作，在实际应用中我们会同时处理若干训练数据，该方法被称为小批量梯度下降法(Mini-Batch Gradient Descent)。

同时，随机梯度下降法和小批量梯度下降法相当于引入了噪声，有利于算法逃出局部最小值点。

小批量梯度下降法如何选取参数m?

在不同的应用中，最优的m通常会不一样，需要通过调参选取。一般m取2的幂次时能充分利用矩阵运算操作，所以可以在2的幂次中挑选最优的取值，例如32、64、128、256等。过小的批量大小难以利用多核架构（分布式）。

小批量梯度下降法如何挑选m个训练数据?

为了避免数据的特定顺序给算法收敛带来的影 响，一般会在每次遍历训练数据之前，先对所有的数据进行随机排序，然后在每 次迭代时按顺序挑选m个训练数据直至遍历完所有的数据。

小批量梯度下降法如何选取学习速率α?

为了加快收敛速率，同时提高求解精度，通常会 采用衰减学习速率的方案:一开始算法采用较大的学习速率，当误差曲线进入平 台期后，减小学习速率做更精细的调整。最优的学习速率方案也通常需要调参才能得到。

小批量梯度下降法的优化?

主要是从学习率和梯度估计两方面进行优化。

从学习率上来看，过大的学习率会导致算法不能收敛，而过小的学习率会导致收敛速度过慢，在不同阶段需要不同大小的学习率。在高维特征数据上，学习率不仅要考虑梯度，还要考虑曲率（二阶梯度）。不同特征的曲率不同，所需要的步长（学习率）也不一样，高曲率的特征方向所需步长较小，低曲率方向所需步长较大。如果各个特征方向都使用同样的学习率，会导致在低曲率方向收敛，而高曲率方向来回震荡。

从梯度估计的角度看，由于深度学习损失函数往往不是凸函数，存在大量的鞍点、局部最小值点，这些点处，梯度几乎为0，如果用经典梯度下降法，可能会陷入局部最小值点或鞍点。

学习率调整有哪些方法?

1. 固定学习率衰减。例如：分段常数衰减、逆时衰减、（自然）指数衰减、余弦衰减
2. 周期性学习率。例如：循环学习率、带热重启的随机梯度下降法
3. 学习率预热。由于参数是随机初始化的，梯度往往较大，所以最开始几轮使用较小的学习率，等梯度下降到一定程度后再恢复。
4. 自适应算法。例如：AdaGrad、AdaDelta、RMSProp、Adam等

梯度估计修正有哪些方法?

1. 动量法（Momentum）。综合考虑了当前梯度和之前梯度的影响，可以通过局部最小值点。
2. Nesterov加速法。是动量法的简单变形
3. Adam。
4. 梯度截断。例如：按值截断，设置一个区间[a, b]，梯度小于a时设为a，大于b时设为b。 按模截

断，给定一个截断阈值b，如果 $||g_t||^2 \leq b$ ，保持梯度不变，如果 $||g_t||^2 > b$ ，令 $g_t = \frac{b}{||g_t||} g_t$

请谈谈你所知道的自适应梯度算法

AdaGrad。AdaGrad的设计出发点是，如果梯度较大，那么所需要的学习率较小。如果梯度较小，所需的学习率较大。它的梯度更新公式为：

$$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\sqrt{\sum_{n=1}^{t} g_i(n)^2 + \epsilon}} g_i(t)$$

相比较经典梯度下降法，我们将学习率从 η 改成了 $\frac{\eta}{\sqrt{\sum_{n=1}^{t} g_i(n)^2 + \epsilon}}$ ，利用前t个时间的梯度平方和来调整学习率的大小。刚开始训练时，学习率较大，随着训练不断进行，学习率越来越小。 ϵ 是一个极小的值，防止分母为0， η 通常取0.01。

缺点：随着学习迭代，分母可能快速增大到一个很大的值，学习速度越来越慢。到一定阶段学习率部分为0，此时梯度还不为0，会导致早停止的问题。（训练没有完成）

AdaDelta。为了解决AdaGrad梯度下降过快可能导致早停止的问题，AdaDelta不再累加过去t时间的梯度平方和，而是求过去所有梯度平方的平均值。AdaDelta不再需要全局的学习率， γ 通常取0.9。

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_i(t)$$

同时对 $\Delta\theta^2$ 参数也进行更新：

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta^2$$

$$\Delta\theta_i(t) = \frac{\sqrt{E[\Delta\theta_i^2]_{t-1} + \epsilon}}{\sqrt{E[g_i^2]_t + \epsilon}} g_i(t)$$

$$\theta_i(t+1) = \theta_i(t) - \Delta\theta_i(t)$$

RMSProp。RMSProp的初衷与AdaDelta类似，为了解决AdaGrad梯度下降过快可能导致早停止的问题，可以看作是AdaDelta的简化版。γ 通常取0.9， η 通常取0.001。

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_i(t)$$

动量法（Momentum）是什么？有什么作用?

Momentum 通过加入 γv_{t-1} ，可以加速 SGD， 并且抑制震荡。momentum即动量，它模拟的是物体运动时的惯性，即更新的时候在一定程度上保留之前更新的方向，同时利用当前batch的梯度微调最终的更新方向。这样一来，可以在一定程度上增加稳定性，从而学习地更快，并且还有一定摆脱局部最优的能力。动量法做的很简单，相信之前的梯度。如果梯度方向不变，就越发更新的快，反之减弱当前梯度。 γ 一般为0.9。γ 取0.9时，相当于加速10倍。

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta_t = \theta_{t-1} - v_t$$

缺点：这种情况相当于小球从山上滚下来时是在盲目地沿着坡滚，如果它能具备一些先知，例如快要上坡时，就知道需要减速了的话，适应性会更好。

Nesterov加速法是什么?

Nesterov加速法与动量法相比，梯度计算不是在当前位置，而是未来的位置上。这个下降更加智能，当梯度方向快要改变的时候，它提前获得了该信息，从而减弱了这个过程，再次减少了无用的迭代。γ 一般为0.9。

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

请谈谈Adam?

Adam = RMSprop + Momentum

$m_t = \gamma_1 m_{t-1} + (1-\gamma_1) g_t$ (Momentum法)

$v_t = \gamma_2 v_{t-1} + (1-\gamma_2) g_t^2$ (RMSprop法)

在实践中发现，当 m_t 和 v_t 初始化为0，且 γ_1 和 γ_2 接近于1时（γ_1 和 γ_2 通常取0.9），m_t 和 v_t 偏向于0，导致在训练初始阶段，学习速度很小，所以对 m_t 和 v_t 做了一个偏差校正：

$$\hat{m}_t = \frac{m_t}{1-\gamma_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\gamma_2^t}$$

梯度更新公式为：$\theta_i(t+1) = \theta_i(t) - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$ ，实践表明，Adam 比其他适应性学习方法效果要好。