刘程华 2018011687 计91

Problem 1

引理：任意 $z\in\mathbb{R}^m$ 有 $\frac{1}{\sqrt{m}}\|z\|_2\le\|z\|_\infty\le\|z\|_2$

证明显然成立：$z^Tz\le m[\max(z_i)]^2$，$[\max(z_i)]^2\le z^Tz$

$\|Ax_{ch}-b\|_\infty\ge\frac{1}{\sqrt{m}}\|Ax_{ch}-b\|_2\ge\frac{1}{\sqrt{m}}\|Ax_{ls}-b\|_2\ge\frac{1}{\sqrt{m}}\|Ax_{ls}-b\|_\infty$

$\Rightarrow\ \|Ax_{ls}-b\|_\infty\le\sqrt{m}\|Ax_{ch}-b\|_\infty$

对于无约束范数逼近问题 minimize $\|Ax-b\|$

其等价于 minimize $\|y\|$ s.t $Ax-b=y$ $\Rightarrow$ 对偶问题 maximize b^Tv s.t $\|v\|_*\le1$, $A^Tv=0$

所以 chebyshev 逼近问题的对偶问题

maximize b^Tv

s.t $\|v\|_1\le1$

$A^Tv=0$

任何可行v是 $\|Ax_{ch}-b\|_\infty$ 的一个下界. b^Tv

定义 $r_{ls}=b-Ax_{ls}$. $A^Tr_{ls}=A^T(b-A(A^TA)^{-1}A^Tb)=0$

$r_{ls}\neq0$ 时，$\hat{v}=\frac{r_{ls}}{\|r_{ls}\|_1}$ 是对偶问题的一个可行解

任意到 $\|x\|_1\le\sqrt{m}\|x\|_2$，$\|x\|_\infty\le\|x\|_2$

$\Rightarrow\ \sqrt{m}\|r_{ls}\|_2^2\ge\|r_{ls}\|_\infty\|r_{ls}\|_\infty\Rightarrow\frac{\|r_{ls}\|_2^2}{\|r_{ls}\|_1}\ge\frac{1}{\sqrt{m}}\|r_{ls}\|_\infty$

$\Rightarrow\ \frac{\|r_{ls}\|_2^2}{\|r_{ls}\|_1}$ 是比 $\|r_{ls}\|_\infty$ 更好的下界

A General and Adaptive Robust Loss Function

刘程华 2018011687 计91

Abstract：

文章对现有的几种损失函数进行了泛用性的推广。通过引入鲁棒性作为连续参数，该损失函数可以使围绕最小化损失的算法得以推广，从而提高基本视觉任务（如配准和聚类）的性能。同时如果将该损失解释为单变量密度的负对数可得出一般的概率分布，其中包括正态分布和柯西分布作为特殊情况。这种概率性解释使得能够进行神经网络的训练，其中损失的鲁棒性在训练过程中自动自我适应，从而提高了基于学习任务的性能，例如生成图像合成和无监督单眼深度估计，而无需任何手动的参数调整。

Loss Function：

$$f(x,\alpha,c)=\frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x/c)^2}{|\alpha-2|}+1\right)^{\frac{\alpha}{2}}-1\right)$$

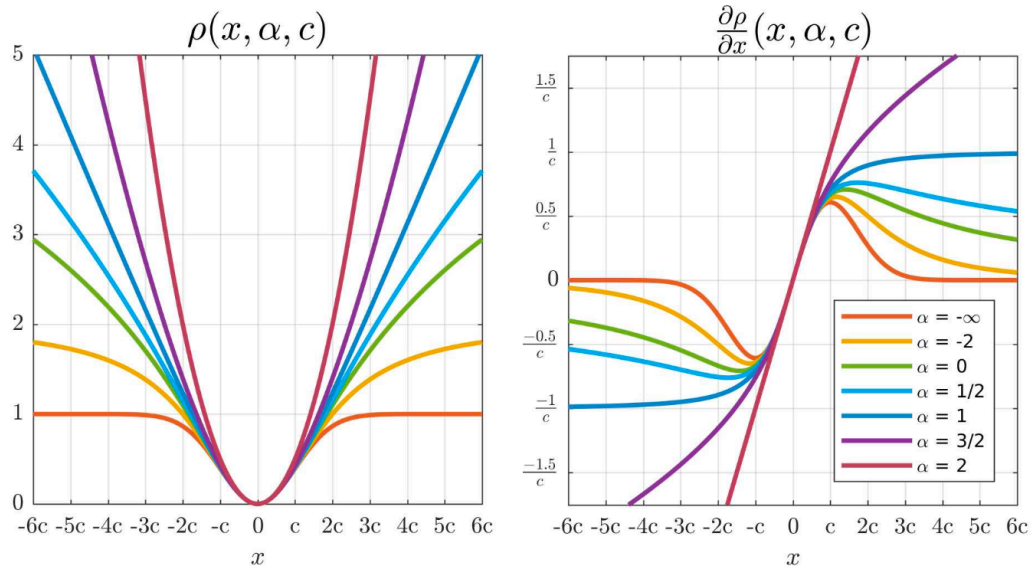
x可以看做期望值与实际值的差值，α控制损失函数的鲁棒性。c可以看作是一个尺度参数，在x=0附近控制弯曲的尺度。由于α是超参数，我们可以看到，对于α的不同值，损失函数具有下述相似的形式： $0.5\left(\frac{x}{c}\right)^2$; for limit $\alpha\to 2$; $L2$ Loss $\sqrt{\left(\frac{x}{c}\right)^2+1}-1$; for $\alpha=1$; L1 Loss. $\log\left(\left(\frac{x}{c}\right)^2+1\right)$; for limit $\alpha\to 0$; Cauchy Loss. $\frac{2\left(\frac{z}{c}\right)^2}{\left(\frac{z}{c}\right)^2+4}$; for $\alpha=-2$; Geman-McClure Loss. $1-\exp\left(-0.5\left(\frac{x}{c}\right)^2\right)$; for limit $\alpha\to-\infty$; Welsch Loss.

损失函数在$\alpha=0$和2处没有定义，但是取极限我们可以进行近似：

$$\rho(x,\alpha,c)=\begin{cases}\frac{1}{2}(x/c)^2 & \text{if } \alpha=2\\ \log\left(\frac{1}{2}(x/c)^2+1\right) & \text{if } \alpha=0\\ 1-\exp\left(-\frac{1}{2}(x/c)^2\right) & \text{if } \alpha=-\infty\\ \frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x/c)^2}{|\alpha-2|}+1\right)^{\alpha/2}-1\right) & \text{otherwise}\end{cases}$$

求导有：

$$\frac{\partial\rho}{\partial x}(x,\alpha,c)=\begin{cases}\frac{x}{c^2} & \text{if } \alpha=2\\ \frac{2x}{x^2+2c^2} & \text{if } \alpha=0\\ \frac{x}{c^2}\exp\left(-\frac{1}{2}(x/c)^2\right) & \text{if } \alpha=-\infty\\ \frac{x}{c^2}\left(\frac{(x/c)^2}{|\alpha-2|}+1\right)^{(\alpha/2-1)} & \text{otherwise}\end{cases}$$



对于所有 α 值, 当 $|x| < c$ 时，导数近似为线性，因此小的残差的影响始终与该残差的大小成线性比例。 如果 $\alpha = 2$, 导数的大小与残差的大小成线性比例关系 — 较大的残差会产生较大的影响。 如果 $\alpha = 1$, 则当 $|x|$ 逐渐大于 c 时，导数的大小将饱和至恒定的 $1/c$, 因此，随着残差的增 加，其作用永远不会减小，但永远不会超过固定量。 如果 $\alpha < 1$, 当 $|x|$ 逐渐大于 c 时，则导数的大小开始减小（ 用 M-estimation 的话来说，导数 又被称为“影响力", 在"下降"），从而即使离群值的残差增加，该异常值在梯度下降期间的影响也较小。 当 α 变得越来越负时，离群值的影响减弱，并且当 α 接近 $-\infty$ 时，其残差大于 $3c$ 的离群值几乎被完全忽略。 我们还可以根据平均值来合理化 α 。因为一组值的经验均值最小化了均值与集合之间的总平方误差，而经验中位数同样最小化了绝对误差，所以将 $\alpha = 2$ 时的损失最小化就等于估计了平均值, 而 $\alpha = 1$ 时的相似度估计中位数。用 $\alpha = -\infty$ 使损失最小化等效于局部模式寻找。可以将这 些范围之间的 α 值视为在估计期间这三种平均值之间的平滑插值。

Probability Density Function：

利用该损失函数，也可以构建一般的概率分布：

$p(x \mid \mu, \alpha, c) = \frac{1}{cZ(\alpha)} \exp(-\rho(x - \mu, \alpha, c))$

$Z(\alpha) = \int_{-\infty}^{\infty} \exp(-\rho(x, \alpha, 1))$

$p(x \mid \mu, \alpha, c)$ 中 $\alpha \geq 0$， $\alpha < 0$时$Z(\alpha)$发散。下面为 α取定时的值。

$$Z(0) = \pi\sqrt{2} \quad Z(1) = 2eK_1(1)$$
$$Z(2) = \sqrt{2\pi} \quad Z(4) = e^{1/4}K_{1/4}(1/4)$$

$K_n(\cdot)$ 是修正的第二类Bessel 函数。 对于任何有理正数α (不包括奇点$\alpha = 2$)

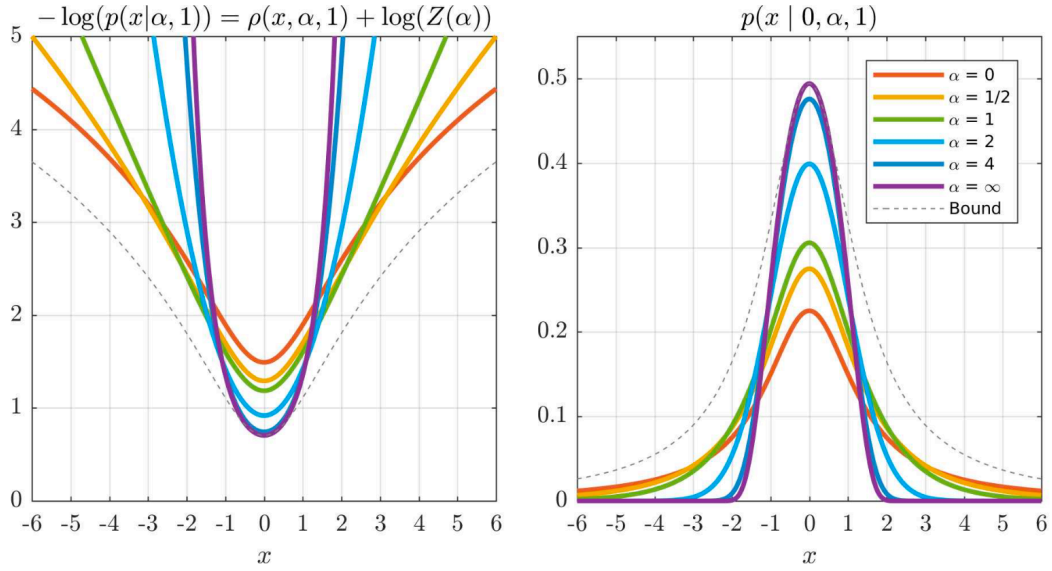
$\alpha = n/d\, n, d \in \mathbb{N}$，我们有：

$$Z\left(\frac{n}{d}\right) = \frac{e^{\frac{2d}{n}-1}\sqrt{\left|\frac{2d}{n}-1\right|}}{(2\pi)^{(d-1)}} G_{p,q}^{0,0}\left(\begin{matrix}\mathbf{a_p} \\ \mathbf{b_q}\end{matrix} \mid \left(\frac{1}{n} - \frac{1}{2d}\right)^{2d}\right)$$

$\mathbf{b_q} = \left\{\frac{i}{n} \mid i = -\frac{1}{2}, \ldots, n - \frac{3}{2}\right\} \cup \left\{\frac{i}{2d} \mid i = 1, \ldots, 2d - 1\right\} \mathbf{a_p} = \left\{\frac{i}{n} \mid i = 1, \ldots, n - 1\right\}$

其中$G(\cdot)$是Meijer G函数，bq是多集（项可能出现两次）。

当 $\alpha = 2$ 时，我们的分布变为正态 (高斯) 分布; 当 $\alpha = 0$ 时，我们的分布变为柯西分布。这也是学生t分布的特殊情况 （分别为 $\nu = \infty$ 和 $\nu = 1$ ），尽管这是这两个分布的族相交的仅有 两点。我们的分布类似于广义的高斯分布，除了它是“平滑的”以便接近原点附近的高斯分布，而与 形状参数 α 无关。下图展示当我们定义损失函数时，极大似然函数的负对数（左）和概率密度（右）。密度受柯西分布的限制。



Experiments：

在实验中，将使用分布 $-\log(p(\cdot \mid \alpha, c))$ 的NLL作为训练我们的神经网络的损失，而不是一般损失 $\rho(\cdot, \alpha, c)$

对于每个输出维度，我们构造无约束的TensorFlow变量$\left\{\alpha_\ell^{(i)}\right\}$ 和并$\left\{c_\ell^{(i)}\right\}$定义

$$\alpha^{(i)} = (\alpha_{\max} - \alpha_{\min})\,\mathrm{sigmoid}\left(\alpha_\ell^{(i)}\right) + \alpha_{\min}$$
$$c^{(i)} = \mathrm{softplus}\left(c_\ell^{(i)}\right) + c_{\min}$$
$$\alpha_{\min} = 0, \alpha_{\max} = 3, c_{\min} = 10^{-8}$$

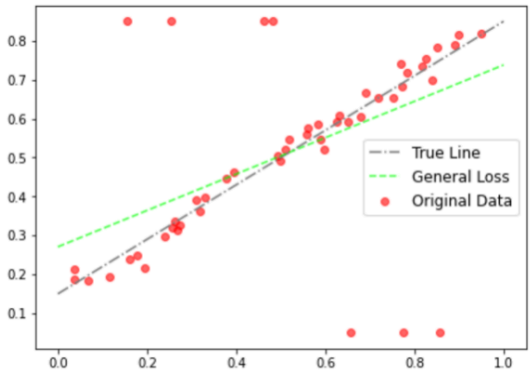
借助Jon Barron 's GitHub存储库中的代码

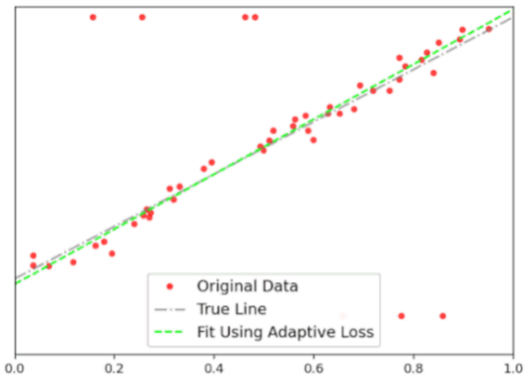
```
https://github.com/jonbarron/robust_loss_pytorch
```

下面创建一个线性数据集（含有正态分布的噪声和离群值）来验证方法的有效性。

```
import numpy as np
import torch

scale_true = 0.7
shift_true = 0.15

x = np.random.uniform(size=n)
y = scale_true * x + shift_true
y = y + np.random.normal(scale=0.025, size=n) # add noise
```

```
flip_mask = np.random.uniform(size=n) > 0.9
y = np.where(flip_mask, 0.05 + 0.4 * (1. - np.sign(y - 0.5)), y)
# include outliers

x = torch.Tensor(x)
y = torch.Tensor(y)

class RegressionModel(torch.nn.Module):

   def __init__(self):
      super(RegressionModel, self).__init__()
      self.linear = torch.nn.Linear(1, 1)
      ## applies the linear transformation.

   def forward(self, x):
      return self.linear(x[:,None])[:,0] # returns the forward pass


###### ex1
##alpha=2
regression = RegressionModel()
params = regression.parameters()
optimizer = torch.optim.Adam(params, lr = 0.01)

for epoch in range(2000):
   y_i = regression(x)
   loss = torch.mean(robust_loss_pytorch.general.lossfun(
     y_i - y, alpha=torch.Tensor([2.]), scale=torch.Tensor([0.1])))
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()


###### ex2
##alpha adaptive
regression = RegressionModel()
adaptive = robust_loss_pytorch.adaptive.AdaptiveLossFunction(
           num_dims = 1, float_dtype=np.float32)
params = list(regression.parameters()) + list(adaptive.parameters())
optimizer = torch.optim.Adam(params, lr = 0.01)
for epoch in range(2000):
   y_i = regression(x)
   loss = torch.mean(adaptive.lossfun((y_i - y)[:,None]))
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()
```





可见adaptive loss更接近真实的线，可以忽略离群值的影响。