信息网络专题研究课机器学习大作业报告（应用层）

一、 文献信息

1、 论文题目：DeepFool: a simple and accurate method to fool deep neural networks

2、 作者：Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard ´ Ecole Polytechnique F´ed´erale de Lausanne

3、 发表途径：fseyed.moosavi,alhussein.fawzi,pascal.frossardg at epfl.ch

4、 发表时间：2016

二、 问题意义

1. 研究背景

FGSM 算法能够快速简单的生成对抗性样例，但是它没有对原始样本扰动的范围进行界定（扰动程度是人为指定的），我们希望通过最小程度的扰动来获得良好性能的对抗性样例。2016 年，Seyed 等人提出的 DeepFool 算法很好的解决了这一问题。

2. 研究问题

文章的核心思想是希望找到一种对抗性扰动的方法来作为对不同分类器对对抗性扰动鲁棒性评估的标准。简单来说就是，现在我需要两个相同任务的分类器 A、B 针对同一个样本生成各自的对抗性样例。对于分类器 A 而言，其生成对抗性样例所需要添加的最小扰动为 a；对于分类器 B 而言，其生成对抗性样例所需要添加的最小扰动为 b；通过对 a、b 的大小进行比较，我们就可以对这两个分类器对对抗性样例的鲁棒性进行评估。由于 FGSM 产生扰动是人为界定的，所以它不能作为评估的依据。DeepFool 可以生成十分接近最小扰动的对抗性样例，因此它可以作为衡量分类器鲁棒性的标准。

三、 思路方法

1. 线性二分类问题

Algorithm 1 DeepFool for binary classifiers

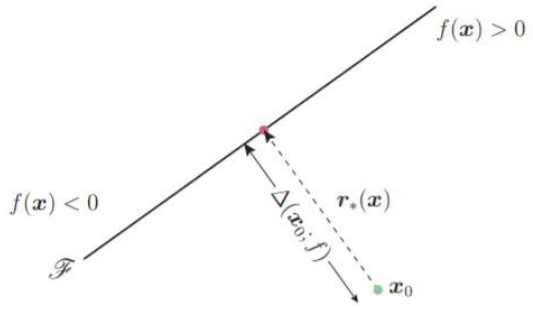
input: Image x, classifier f.
output: Perturbation $\hat{r}$.
Initialize $x_0 \leftarrow x, i \leftarrow 0$.
while $\text{sign}(f(x_i)) = \text{sign}(f(x_0))$ **do**
  $r_i \leftarrow -\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2}\nabla f(x_i)$,
  $x_{i+1} \leftarrow x_i + r_i$,
  $i \leftarrow i + 1$.
end while
return $\hat{r} = \sum_i r_i$.

DeepFool 源于对分类问题的思考。对于如图所示的线性二分类问题，令$f(x) = w^T +$

b，其中$F = \{x: f(x) = 0\}$。此时位于直线的下方，即$f(x_0) > 0$。现在我们希望对x_0添加扰动r，使得分类器$f(x_0 + r) < 0$。那么如何添加扰动才能使得扰动的程度最小呢？这个问题可以转化为求点到直线之间的距离，我们通过x_0做直线F的垂线，与F相交于p（投影点），则p与x_0之间的距离即为x_0到分类边界F的最短距离。所以当我们沿着分类边界法线方向对x_0进行扰动，可以保证扰动的程度最小。

$$r(x_0) := \arg\min_k \|r\|_2 = -\frac{f(x_0)}{\|w\|_2^2} w$$

添加扰动之后将 x_0 映射到分类边界的投影点 p，即 p=x_0+r(x_0)。



同样，对于非线性的二分类问题（分类边界为曲线或者曲面），我们也需要计算从目标样本点到分类边界的最短距离。这个计算过程较为复杂，一般采用垂直逼近法来逐步的逼近x_0在分类边界上的投影点，所以在论文中算法 1 会有一个迭代过程。值得注意的是，我们得到最终$\sum_i r_i$的表示的是从x_0到分类边界投影点的距离向量，满足$f(x_0 + \sum_i r_i) = 0$。如果要使分类结果改变，需要再添加一些极小的扰动，如$f\big(x_0 + (1 + \eta)\sum_i r_i\big) < 0$（文中η取 0.02）。

2. 多分类任务

Algorithm 2 DeepFool: multi-class case

input: Image x, classifier f.
output: Perturbation $\hat{r}$.

Initialize $x_0 \leftarrow x$, $i \leftarrow 0$.
while $\hat{k}(x_i) = \hat{k}(x_0)$ **do**
  for $k \neq \hat{k}(x_0)$ **do**
    $w'_k \leftarrow \nabla f_k(x_i) - \nabla f_{\hat{k}(x_0)}(x_i)$
    $f'_k \leftarrow f_k(x_i) - f_{\hat{k}(x_0)}(x_i)$
  end for
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(x_0)} \frac{|f'_k|}{\|w'_k\|_2}$
  $r_i \leftarrow \frac{|f'_{\hat{l}}|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$
  $x_{i+1} \leftarrow x_i + r_i$
  $i \leftarrow i + 1$
end while
return $\hat{r} = \sum_i r_i$

对于多分类任务，思路也大致相同。在线性多分类任务中，需要注意的大致有两点：

首先，对于多分类任务的分类边界要重新进行定义。我们令f(x)为分类器，$\hat{k}(x) = arg\max_k f_k(x)$表示其对应的分类结果（一共有 k 个类别）。分类边界定义为：

$$\mathscr{F}_k = \{x : f_k(x) - f_{\hat{k}(x_0)}(x) = 0\}$$

其次，由于分类边界有多个，我们需要以此求出x_0到每个分类边界的距离（类似于进行多次二分类的计算过程），然后进行比较，选择其中最短距离向量作为最终的扰动。

$$\hat{l}(x_0) = \arg\min_{k \neq \hat{k}_{x_0}} \frac{|f_k(x_0) - f_{\hat{k}(x_0)}(x_0)|}{\|w_k - w_{\hat{k}_{x_0}}\|_2}$$

之后通过计算r(x_0)得到x_0在分类边界上的投影。

在非线性多分类任务中，与线性多分类的区别在于其分类边界是不确定的，所以我们需要采用类似于非线性二分类任务中的方法来逼近分类边界。获得分类边界之后的计算与线性多分类的过程类似。此后，重复该过程多次，即可获得最终x_0在分类边界的投影。

四、 代码实现

1. 网络模型

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 300)
        self.fc2 = nn.Linear(300, 100)
        self.fc3 = nn.Linear(100, 10)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
    return x
```

2. 数据集

```
# 定义数据转换格式
mnist_transform = transforms.Compose([transforms.ToTensor(),
```

```
transforms.Lambda(lambda x : x.resize_(28*28))])
    # 导入数据，定义数据接口
    testdata  = torchvision.datasets.MNIST(root="./mnist", train=False, download=True, transform=mnist_transform)
testloader  =  torch.utils.data.DataLoader(testdata,  batch_size=256, shuffle=True, num_workers=0)
```

3. 实现

以下讨论多分类情况下 DeepFool 算法的代码实现，模型结构以及数据集等与上述一致，不再重复。首先，我们选择一个测试样本，生成该样本的对抗性样例。

```
net = torch.load('mnist_net_all.pkl') # 加载模型
    index = 100 # 选择测试样本
    image=Variable(testdata[index][0].resize_(1,784),requires_grad=True)
label = torch.tensor([testdata[index][1]])
```

接下来，通过前向传播的过程，我们获得该样本对每一类别可能的取值情况，并将其从大到小排列起来，列表 I 对应其索引值，label 即 $\widehat{k_{x_0}}$。

```
    f_image = net.forward(image).data.numpy().flatten()
    # f_image: [-1.1256416 , -1.0344085 ,  2.0596995 , -2.0181773 , -0.24274658, -0.53373957,  6.6361637 , -2.250309  ,  0.06580263, -3.0854702 ]

    I = (np.array(f_image)).flatten().argsort()[::-1]
    # I: [6, 2, 8, 4, 5, 1, 0, 3, 7, 9]
label = I[0] # 该样本的标签为 6
```

然后，我们定义一些需要用到的变量

```
    input_shape = image.data.numpy().shape # 获取原始样本的维度
    pert_image = copy.deepcopy(image) # 深度复制原始样本
    w = np.zeros(input_shape)
    r_tot = np.zeros(input_shape)

    loop_i = 0
    max_iter = 50 # 最多迭代次数
```

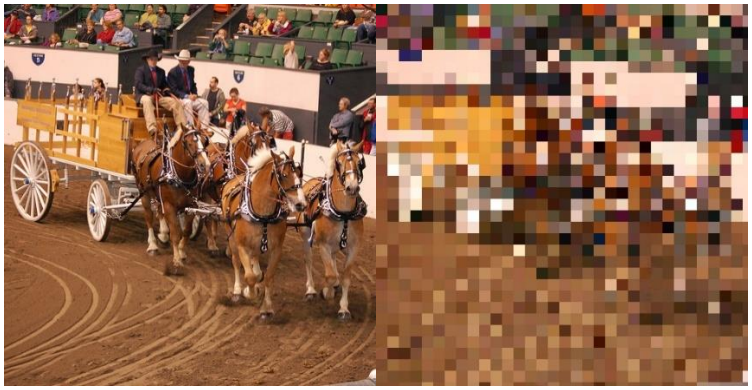
```
    overshoot = 0.02
    x = Variable(pert_image, requires_grad=True)
    fs = net.forward(x)
    fs_list = [fs[0][I[k]] for k in range(len(I))] # 每个类别的取值情况，及其对应的梯度值
k_i = label
```

下面是算法实现的核心部分，参考论文中的伪代码，其中 orig_grad 表示$\nabla f_{\widehat{k_{x_0}}}(x_i)$，cur_grad 表示$\nabla f_k(x_i)$，fs[0][I[k]]表示$f_k(x_i)$，fs[0][I[0]]表示$f_{\widehat{k_{x_0}}}(x_i)$。通过内部的 for 循环可以获得 x 到各分类边界的距离；在外部的 while 循环中，我们利用内部循环获得的所有边界距离中的最小值对 x 进行更新。重复这一过程，直到的分类标签发生变化。

```
    while k_i == label and loop_i < max_iter:
        pert = np.inf
        fs[0][I[0]].backward(retain_graph=True)
        orig_grad = x.grad.data.numpy().copy()

        for k in range(len(I)):
            zero_gradients(x)
            fs[0][I[k]].backward(retain_graph=True)
            cur_grad = x.grad.data.numpy().copy()

            w_k = cur_grad - orig_grad
            f_k = (fs[0][I[k]] - fs[0][I[0]]).data.numpy()

            pert_k = abs(f_k) / np.linalg.norm(w_k.flatten())
            if pert_k < pert: # 获得最小的分类边界距离向量
                pert = pert_k
                w = w_k
        r_i = (pert + 1e-4) * w / np.linalg.norm(w)
        r_tot = np.float32(r_tot + r_i) # 累积扰动

        pert_image = image + (1+overshoot)*torch.from_numpy(r_tot) # 添加扰动
```

```
        x = Variable(pert_image, requires_grad=True)
        fs = net.forward(x)
        k_i = np.argmax(fs.data.numpy().flatten()) # 扰动后的分类标签
        loop_i += 1
r_tot = (1+overshoot)*r_tot # 最终累积的扰动
```

对原始图片，添加扰动获得如下图片



五、 总结

本文中,作者基于迭代且线性近似的方案提出了一种计算对抗样本的方法 DeepFool。DeepFool 方法产生的对抗样本是几乎不可察觉的，大量的实验结果表明，这是一种非常精确且有效的对抗扰动计算方法，它可以提供一个有效的方式去评估分类器的鲁棒性并且可以通过恰当的微调改善分类器的表现。此外，利用 DeepFool 算法产生的对抗样本做对抗训练也可以改善神经网络的正确率。这对于图像识别工作是一项巨大的贡献。