学习：Deep Leakage from Gradients

相关链接：

1. Deep Leakage from Gradients原文
2. GitHub上的论文源码

1. 摘要、介绍

1.1 这篇文章在什么背景下研究了什么问题?

这篇文章的标题是 "Deep Leakage from Gradients"，作者是 Ligeng Zhu, Zhijian Liu, Song Han。文章的背景和研究问题主要涉及以下几个方面：

1. 分布式训练：训练大型机器学习模型（例如深度神经网络）是计算密集型的。为了在合理的时间内完成训练过程，许多研究都在分布式训练上进行了工作以加速。分布式训练可以分为两类：有参数服务器（集中式）和无参数服务器（去中心化）。
2. 隐私敏感的训练数据：在许多应用场景中，训练数据是隐私敏感的。例如，患者的医疗状况不能在医院之间共享。为了避免敏感信息被泄露，最近出现了协作学习，两个或更多的参与者可以共同训练一个模型，而训练数据集永远不会离开每个参与者的本地服务器。只有梯度在网络中共享。
3. 从梯度中的“浅”泄漏：以前的工作已经对如何从梯度中推断训练数据的信息进行了一些探索。对于一些层，梯度已经泄漏了某种级别的信息。

文章的主要贡献包括：

1. 作者们展示了从公开共享的梯度中获取私有训练数据的可能性。据他们所知，这是第一个实现这一点的算法，名为深度梯度泄漏（Deep Leakage from Gradients，简称DLG）。
2. DLG只需要梯度就可以揭示像素精确的图像和匹配的文本。而传统的方法通常需要额外的信息来进行攻击，并且只能产生部分属性或合成的替代品。
3. 为了防止重要数据的潜在泄漏，他们分析了在各种设置中的攻击难度，并讨论了针对攻击的几种防御策略。

这篇文章的研究背景和问题主要集中在分布式训练和数据隐私保护的问题上，特别是在梯度共享的情况下如何保护训练数据的隐私。他们提出了一个新的算法DLG，可以从梯度中获取私有训练数据，并讨论了防止数据泄漏的策略。

1.2 文中提到“shallow” leakages是什么？

"Shallow" leakages是指从梯度中推断出训练数据的某些属性的现象。这种泄漏通常需要额外的标签信息，并且只能生成与训练图像相似的合成图像。这种泄漏被认为是"浅"的，因为它并不能完全揭示训练数据的所有信息。

然而，这篇文章中的作者提出了一种新的"深度"泄漏（Deep Leakage from Gradients，DLG）方法，它可以从梯度中获取私有训练数据。DLG是一个优化过程，不依赖于任何生成模型，因此不需要关于训练集的任何额外先验知识。相反，它可以从共享的梯度中推断出标签，而DLG产生的结果（图像和文本）是精确的原始训练样本，而不是合成的类似替代品。

总的来说，"shallow" leakages是指从梯度中获取训练数据的部分信息，而"deep" leakages则可以获取更多的训练数据信息。

1.3 文中提出了什么方法可以抵御Deep Leakage from Gradients？

文章中提出了几种可能的防御策略来抵御Deep Leakage from Gradients（DLG）：

梯度修剪：在不改变训练设置的情况下，最有效的防御方法是梯度修剪。

梯度压缩：压缩梯度也是一种实用的防御方法。当修剪的梯度超过20%时，DLG的攻击就会被有效防止。

增大批量大小：如果允许改变训练设置，那么有更多的防御策略。增大批量大小会使泄漏更难发生，因为在优化过程中需要解决更多的变量。

提高输入图像的分辨率：按照这个思路，提高输入图像的分辨率也可以是一种好的防御方法，尽管这需要对CNN架构进行一些改变。

使用密码学：密码学也可以用来防止泄漏：Bonawitz等人设计了一个安全聚合协议，Phong等人提出在发送之前加密梯度。在所有的防御中，密码学是最安全的一种。然而，这两种方法都有其局限性，不够通用：安全聚合要求梯度是整数，因此与大多数CNN不兼容，而同态加密只针对参数服务器。

这些策略的有效性可能会受到训练设置和数据特性的影响，因此在实际应用中可能需要根据具体情况进行调整。

1.4 为什么对全连接层梯度更新的观察可以用来推断输出的特征值，但不能扩展到卷积层呢？

在全连接层中，梯度更新的观察可以用来推断输出特征值。这是因为全连接层的每个神经元都与前一层的所有神经元相连，因此每个神经元的梯度都包含了前一层所有神经元的信息。通过观察这些梯度，我们可以推断出前一层神经元的输出特征值。

然而，这种方法不能扩展到卷积层。卷积层的神经元只与前一层的一小部分神经元相连，这些神经元位于前一层的一个小的局部区域（也就是卷积核的覆盖范围）内。因此，卷积层的每个神经元的梯度只包含了前一层局部区域内神经元的信息，而不是前一层所有神经元的信息。这使得我们无法通过观察卷积层的梯度来推断出前一层的输出特征值。

此外，卷积层的特征图的大小通常远大于权重的大小。这意味着，即使我们可以从卷积层的梯度中获取一些信息，也无法获取足够的信息来完全推断出前一层的输出特征值。

这就是为什么全连接层的梯度更新可以用来推断输出特征值，但不能扩展到卷积层的原因。

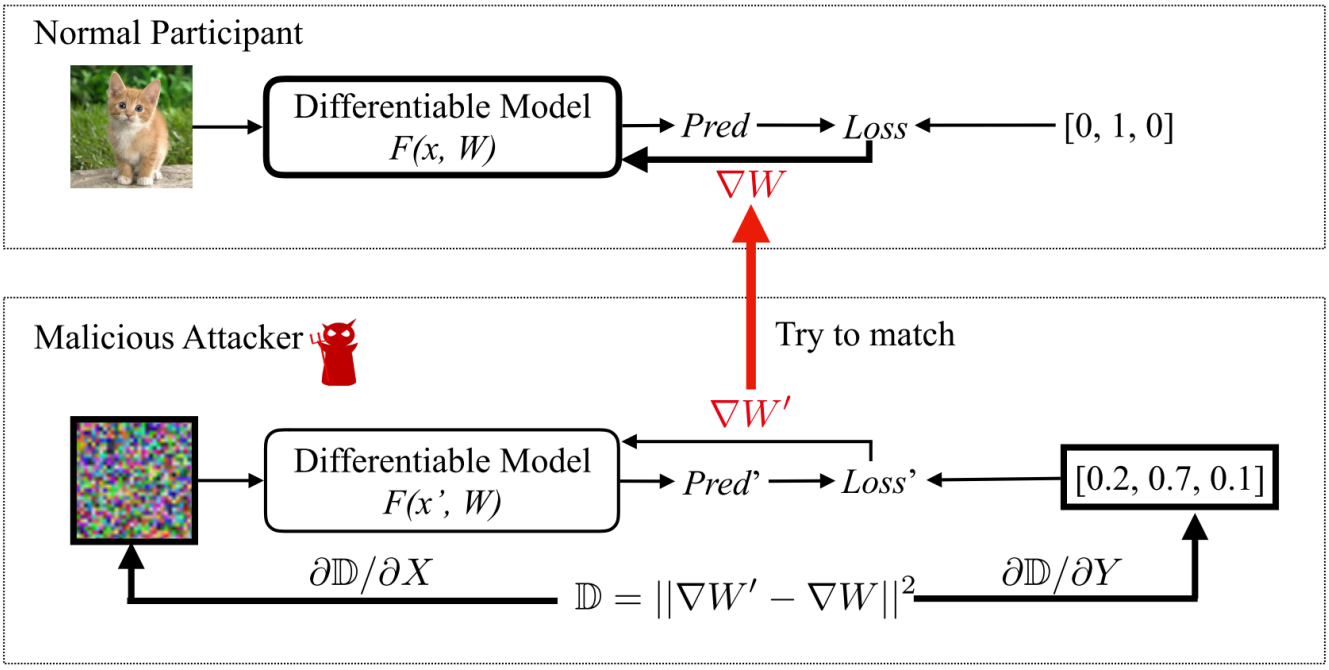
举个例子，假设我们有一个全连接层，它有10个输入节点和1个输出节点。当我们观察到这个全连接层的梯度更新时，我们可以看到所有10个输入节点对输出节点的影响。而如果我们有一个10x10的输入特征图和一个3x3的卷积核，那么输出特征图的大小将会是8x8（假设我们没有使用填充，并且步长为1）。每个输出特征图的像素点是由输入特征图的一个3x3的局部区域的加权和得到的。这意味着，每个输出特征图的像素点只与输入特征图的一小部分像素点有关，而不是所有像素点。因此，当我们观察到卷积层的梯度更新时，我们只能看到输入特征图的局部区域对输出特征图的影响，而不能看到所有输入像素点的影响。这就是为什么我们不能通过观察卷积层的梯度更新来推断输出特征值的原因。

> （但是为什么不能看到所有输入像素点的影响，就不能通过观察卷积层的梯度更新来推断输出特征值的原因呢？？）

1.5 为什么使用GAN攻击的效果是有限的?

2. DLG方法

2.1 DLG算法的工作流程



DLG（Deep Leakage from Gradients）算法的步骤如下：

初始化：首先，算法初始化虚拟的输入和标签。这些虚拟的输入和标签是随机生成的，并且在算法的后续步骤中会被更新。

计算虚拟梯度：在每一步中，算法计算虚拟输入和标签的梯度。这些梯度是通过对机器学习模型的损失函数进行微分得到的。

计算梯度距离：然后，算法计算虚拟梯度和真实梯度之间的距离。这个距离是通过计算虚拟梯度和真实梯度之间的欧几里得距离得到的。

更新虚拟输入和标签：接着，算法更新虚拟输入和标签，以使虚拟梯度更接近真实梯度。这个更新是通过梯度下降算法实现的，其中学习率η是一个超参数。

重复上述步骤：算法重复上述步骤，直到达到预定的迭代次数，或者虚拟梯度和真实梯度之间的距离小于某个阈值。

通过这个过程，DLG算法能够从公开共享的梯度中获取私有的训练数据。

2.2 为什么需要假设F是二次可微的?

3. 实验

3.1 文章的第4.3节中，作者对算法的第六行进行了修改。修改后的公式有什么含义?

Algorithm 1 Deep Leakage from Gradients.

Input: $F(\mathbf{x}; W)$: Differentiable machine learning model; W: parameter weights; ∇W: gradients calculated by training data

Output: private training data $\mathbf{x}, \mathbf{y}$

procedure DLG(F, W, ∇W)
  $\mathbf{x'}_1 \leftarrow \mathcal{N}(0,1)$, $\mathbf{y'}_1 \leftarrow \mathcal{N}(0,1)$ ▷ Initialize dummy inputs and labels.
  for $i \leftarrow 1$ to n **do**
    $\nabla W'_i \leftarrow \partial \ell(F(\mathbf{x'}_i, W_t), \mathbf{y'}_i)/\partial W_t$ ▷ Compute dummy gradients.
    $\mathbb{D}_i \leftarrow ||\nabla W'_i - \nabla W||^2$
    $\mathbf{x}'_{i+1} \leftarrow \mathbf{x}'_i - \eta \nabla_{\mathbf{x}'_i} \mathbb{D}_i$, $\mathbf{y}'_{i+1} \leftarrow \mathbf{y}'_i - \eta \nabla_{\mathbf{y}'_i} \mathbb{D}_i$ ▷ Update data to match gradients.
  end for
  return $\mathbf{x}'_{n+1}, \mathbf{y}'_{n+1}$
end procedure

原公式：

$$\begin{aligned} \mathbf{x}'_{i+1} &\leftarrow \mathbf{x}'_i - \eta \nabla_{\mathbf{x}'_i} \mathbb{D}_i, \\ \mathbf{y}'_{i+1} &\leftarrow \mathbf{y}'_i - \eta \nabla_{\mathbf{y}'_i} \mathbb{D}_i \end{aligned} \tag{1}$$

在算法的第六行中，作者对批处理数据进行了修改。在原始算法中，当批处理数据的大小N≥1时，如果我们直接应用算法，算法的收敛速度会很慢。作者认为这是因为批处理数据可以有N!种不同的排列，这使得优化器难以选择梯度方向。

为了使优化更接近解决方案，作者选择更新一个单独的训练样本，而不是更新整个批处理。因此，他们修改了算法的第六行，修改后的公式：

$$\begin{aligned}\mathbf{x}'^{i \bmod N}_{t+1} &\leftarrow \mathbf{x}'^{i \bmod N}_{t} - \nabla_{\mathbf{x}'^{i \bmod N}_{t+1}} \mathbb{D} \\ \mathbf{y}'^{i \bmod N}_{t+1} &\leftarrow \mathbf{y}'^{i \bmod N}_{t} - \nabla_{\mathbf{y}'^{i \bmod N}_{t+1}} \mathbb{D}\end{aligned} \tag{2}$$

这里的x′和y′分别代表虚拟输入和标签，N是批处理的大小，i是迭代次数，t是时间步，D是梯度距离，∇是梯度符号。

这个修改的含义是，对于每一次迭代，我们只更新一个训练样本（即i mod N），而不是更新整个批处理。我们通过减去对应的梯度距离的梯度来更新这个训练样本。这样做可以使优化更快地收敛，并且可以观察到稳定的收敛性。