Kitsune:在线网络入侵检测自动编码器集成

伊索尔·米尔斯基，托梅尔·多伊茨曼，尤瓦尔·埃洛维奇和阿萨夫·沙卜泰
内盖夫本古里安大学
{yisroel，tomerdoi}@post.bgu.ac.il，{elovici，shabtaia}@bgu.ac.il

摘　要

神经网络已经成为网络入侵检测系统越来越普遍的解决方案(NIDS)。他们学习复杂模型和行为的能力使他们成为区分正常流量和网络攻击的合适解决方案。然而，神经网络的一个缺点是需要大量的资源来训练它们。许多网络网关和路由器设备，可能是 NIDS 的主机，根本没有内存或处理能力来训练，有时甚至执行这样的模型。更重要的是，现有的神经网络解决方案是以监督的方式训练的。这意味着专家必须标记网络流量，并不时手动更新模型。

在本文中，我们介绍了 Kitsune：一个即插即用的 NIDS，它可以在没有监督的情况下，以高效的在线方式学习检测对本地网络的攻击。Kitsune 的核心算法(KitNET)使用一个称为自动编码器的神经网络集合来共同区分正常和异常的流量模式。KitNET 由一个特征提取框架支持，该框架有效地跟踪每个网络通道的模式。我们的评估显示，Kitsune 可以检测各种攻击，其性能可与离线异常检测器媲美，即使是在树莓派上。这表明 Kitsune 可以成为一个实用而经济的 NIDS。

关键词: 异常检测；网络入侵检测；在线算法；自动编码器；集成学习

一、导言

多年来，针对计算机网络的攻击数量一直在增加[1]。网络入侵检测系统(NIDS)是一种常用的网络安全系统。NIDS 是一种设备或软件，它用于监控所有通过某个策略点的流量，以防止恶意活动。当检测到这样的活动时，会生成一个警报，并发送给管理员。通常，NIDS 部署在单点，例如互联网网关。此点部署策略可以检测进出网络的恶意流量，但不能检测通过网络本身的恶意流量。为了解决这个问题，可以使用分布式部署策略，其中多个 NIDSs 连接到网络中的一组策略路由器和网关。

在过去十年中，人们已经提出了许多机器学习技术来提高检测性能[2][3][4]。一种常见的方法是使用人工神经网络（ANN）来进行网络流量检测。使用人工神经网络的好处是人工神经网络擅长学习输入数据中复杂的非线性概念。这使得人工神经网络相对于其他机器学习算法在检测性能上具有很大优势[5][2]。

使用人工神经网络作为网络入侵检测的普遍方法是训练它将网络流量分类为正常或某种恶意类型的攻击[6]，[7]，[8]。以下展示了在点部署策略中使用基于人工神经网络的分类器的典型方法:

（1）让专家收集包含正常流量和网络攻击的数据集。

（2）使用强大的 CPU 或 GPU 训练人工神经网络来区分正常流量和攻击流量。

（3）向网络/组织的 NIDS 拷贝一份经过培训的模型。

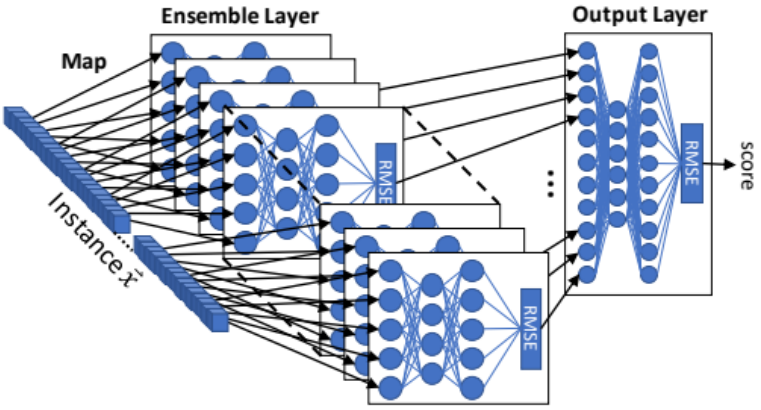
（4）使用 NIDS 对观测到的网络流量执行经过训练的模型。

一般来说，分布式部署策略只有在 NIDSs 的数量能够根据网络规模的大小经济地扩展时才是实用的。实现这一目标的一种方法是将 NIDSs 直接嵌入到廉价的路由器(即使用简单的硬件)。我们认为在这种方法中使用基于人工神经网络的分类器是不切实际的，原因如下:

离线处理。为了训练监督模型，所有标记的实例必须在本地可用。这在简单的网络网关上是不可行的，因为一小时的流量可能包含数百万个数据包。一些工作建议将数据卸载到远程服务器上进行模型培训[9] [3]。但是，这种解决方案可能会产生大量网络开销，并且不具有可伸缩性。

监督学习。贴标签过程耗时且昂贵。更重要的是，什么被认为是正常的取决于 NIDS 观测到的当地流量。此外，随着时间的推移，在攻击不断变化的同时，还会不断发现新的攻击[10]，因此持续维护恶意攻击流量存储库可能是不切实际。最后，分类是一种封闭的识别概念的方法。换句话说，分类器被训练来识别训练集中提供的类。然而，假设所有可能的恶意流量类都可以被收集并将其放入训练数据中是不合理的。

高度复杂。人工神经网络的计算复杂性。



图一：一个 Kitsune 的异常检测算法例子。

随着神经元数量呈指数级增长[11]。这意味着部署在简单网络网关上的人工神经网络在其体系结构和可使用的输入特征数量方面受到限制。这在处理高速流量的网关上是一个很大的问题。

鉴于上述挑战，我们建议开发一种基于人工神经网络的网络入侵检测器，该检测器以分布式方式在路由器上部署和培训，应遵守以下限制:

在线处理。在用实例训练或执行模型之后，实例会立即被丢弃。实际上，可以在任何给定时间内存储少量实例，如在流聚类[12]中所做的那样。

无监督学习。标签明确指出数据包是恶意的还是良性的，但在在训练过程中不被使用。选择使用其他元信息，使得获取信息不会延迟进程。

低复杂性。包处理速率必须超过预期的最大包到达速率。换句话说，我们必须确保没有等待模型处理的数据包队列。

在本文中，我们提出了一种新的基于人工神经网络的 NIDS，它是在线的，无监督的，高效的。在日本民间传说中，Kitsune 是一种神话中的狐狸模样的动物，有许多尾巴，可以模仿不同的形态，其力量随着经验的增加而增加。类似地，Kitsune 有一个小型神经网络(自动编码器)，它们被训练来模拟(重建)网络流量模式，并且它们的性能随着时间的推移而逐渐提高。

Kitsune 异常检测算法（KitNET）的架构如图 1 所示。首先，将实例的特征映射到集合中的可见神经元。接下来，每个自动编码器尝试重建实例的特征，并根据均方根误差(RMSE)计算重建误差。最后，RMSEs 被转发到一个输出自动编码器，该编码器作为一个集成的非线性投票机制。我们注意到在训练 Kitsune 时，一次只有一个实例存储在内存中。KitNET 有一个主要参数，它是集合中任何给定自动编码器的最大输入数。该参数用于提高算法速度，同时在检测性能上适度的权衡。

我们使用自动编码器的原因是因为：(1)它们可以在无监督的方式下进行训练，(2)它们可以在重建效果不佳的情况下用于异常检测。我们建议使用小型自动编码器集成的原因是，在相同的特征空间中，它们比单个自动编码器更高效，噪音更小。从我们的实验中，我们发现 Kitsune 可以将包处理速率提高五倍，并提供与离线(批处理)异常检测器相媲美的检测性能。综上所述，本文的贡献如下：

- 一种新型的基于自动编码器的简单网络设备的 NIDS (Kitsune)，它是轻量级的，即插即用的。据我们所知，我们是第一个提出使用无论带或不带集成电路的自动编码器在计算机网络中进行在线异常检测的。我们还将核心算法(KitNET)作为一种通用的在线无监督异常检测算法，并提供了源代码供下载 [1]。
- 一个用于动态维护和从网络流量中提取隐含的上下文特征的特征提取框架。该框架内存占用很小，因为统计数据是在阻尼窗口上增量更新的。
- 一种自动构建自动编码器集合的在线技术，以无监督的方式将特征映射到人工神经网络输入。该方法涉及特征空间的增量分层聚类(无界数据集的转置)和聚类大小的边界。
- 在一个可操作的网络摄像机视频中监测到一些在网络上的物联网网络和各种攻击的实验结果。我们还通过在树莓派上执行基准测试，演示了该算法在简单路由器上运行的效率和能力。

论文的其余部分组织如下：第二部分讨论了在线异常检测领域的相关工作。第三部分提供了自动编码器的背景知识及其工作原理。第四部分介绍了 Kitsune 的框架和整个机器学习管道。第五部分介绍了检测性能和运行性能方面的实验结果。最后，在第七部分中，我们提出了我们的结论。

二. 相关工作

在过去, 使用机器学习(特别是异常检测)来实现网络入侵检测系统得到广泛研究[13], [14], [15], [16], [17]。然而，这些解决方案通常是对进行训练或执行模型的机器资源没有任何假

设，因此在简单的网关上进行训练和运行的成本太高，或者是需要标记的数据集来执行训练过程。

之前的一些工作已经提出了使用不同轻量级算法的在线异常检测机制。例如，数据包内容[18]或 KNN 算法[19]中的模拟简单直方图的 PAYL 入侵检测系统。这些方法要么非常简单，因此产生非常差的结果，要么需要为训练或检测而积累数据。

网络入侵检测的一种常用算法是人工神经网络。这是因为它能够学习复杂的概念，以及来自网络通信领域的概念[17]。在[20]中，作者在网络入侵检测任务中评估了其他分类算法中的人工神经网络，并提出了一种基于基于连接特征的分类器集成的解决方案。在[8]中，作者提出了对反向传播算法的修改，以提高人工神经网络训练速度。在[7]中，作者使用了基于人工神经网络的多分类器，每个都经过训练以检测特定类型的攻击。在[9]中，作者提出了一种分层技术，其中每个数据包首先通过异常检测模型，如果出现异常，则数据包由一组人工神经网络分类器进行评估，其中每个分类器都被训练来检测特定的攻击类型。

前面提到的所有使用人工神经网络的论文，要么是受监督的，要么不适合简单的网络网关。此外，一些工作假设训练数据可以被存储和累积，而简单的网络网关则不是这样。我们的解决方案支持即插即用部署，这种部署的运行速度比上述模型快得多。

关于自动编码器的使用：在 [21]中，作者使用了一个深度神经网络的集合来处理在线环境中的目标跟踪问题。他们提出的方法使用堆叠的去噪自动编码器(SDAE)。SDAE 的每一层作为原始图像数据的不同特征空间。该方案将 SDAE 的每一层转换为一个用于判别二进制分类器的深度神经网络，。虽然作者在在线环境中使用了自动编码器，但是他们没有执行异常检测，也没有解决实时处理的挑战(这对深度神经网络来说是一个巨大的挑战)。此外，训练深度神经网络是复杂的，并且实际上不能在简单的网络设备上执行。在[22]和[23]中，作者建议使用自动编码器从数据集提取特征，以改进对网络威胁的测试。然而，自动编码器本身并没有用于异常检测。最终，作者使用分类器来检测网络威胁。因此，他们的解决方案需要专家来标记实例，而我们的解决方案是无监督的，即插即用的。

在[24]中，作者提出了用自动编码器检测异常的通用方法。在[19]，作者使用自动编码器来检测电网中的异常。这些成果与我们的不同之处在于，(1)它们不是在线的，(2)作者使用的体系结构不是轻量级的和可扩展的集成体系结构，(3)没有应用到网络入侵检测。我们注意到本文的部分贡献是一个适当的特征提取框架，它允许在在线网络设置中使用自动编码器。

三. 背景：自动编码器

自动编码器是 Kitsune 的基础模块。在本节中，我们将简要介绍自动编码器；他们是什么，以及他们是如何工作的。为了描述自动编码器的训练和执行过程，我们将参考图 2 中的例子。

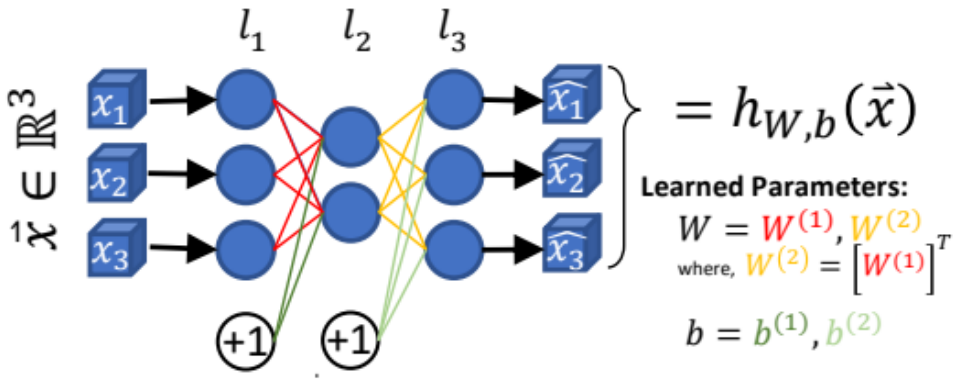


图 2:具有一个压缩层的自动编码器示例，其重建实例具有三个特征。

A. 人工神经网络

人工神经网络由神经元层组成，每层神经元通过突触依次相连。突触有相关权重，这些权重共同定义了模型学习的概念。具体地，设 $l^{(i)}$表示第 i 层，并且让设$|l^{(i)}|$表示 $l^{(i)}$中神经元的数量。最后，让人工神经网络中的总层数表示为 L。将 $l^{(i)}$连接到 $l^{(i+1)}$的权重表示为$|l^{(i)}| \times |l^{(i+1)}|$，矩阵 $W^{(i)}$与$|l^{(i+1)}|$维度偏置向量$\vec{b}^{(i)}$。最后，我们将所有参数 θ 的集合表示为元组 θ≡(W，b)，其中 W 和 b 分别是每层的权重。图 2 说明了神经网络中每一层的突触的权重是如何形成的。

人工神经网络有两层：可视层和隐层。可视层接收带有附加偏置变量(常数值 1)的输入实例$\vec{x}$。$\vec{x}$是描述该实例的数值特征向量，通常标准化大约在[-1、+1]区间内(例如，使用 0-1 标准化或 zscore 归一化)[25]。可视层和隐层之间的区别在于，可视层被认为是预先计算的，并准备传递到第二层。

B. 执行人工神经网络

为了执行人工神经网络，输出值 $l^{(1)}$(即$\vec{x}$)和 $W^{(1)}$加权得出 $l^{(2)}$，然后输出 $l^{(2)}$，并用其与 $W^{(2)}$加权得到 $l^{(3)}$，依此类推，直到最后一层数值得出。这个过程被称为前向传播。设$\vec{a}^{(i)}$为中 $l^{(i)}$神经元中$|l^{(i)}|$输出的向量。为了得到 $a^{(i+1)}$，我们通过计算将$\vec{a}^{(i)}$传递到 $l^{(i+1)}$

$$a^{(i+1)} = f\left(W^{(i)} \cdot \vec{a}^{(i)} + \vec{b^{(i)}}\right) \qquad (1)$$

其中 f 是神经元的激活函数。我们在 Kitsune 中使用的一个常见的激活函数是 sigmoid 函数，其定义为

$$f(\vec{x}) = \frac{1}{1 + e^{\vec{x}}} \qquad (2)$$

最后，我们将最后一层的输出定义为$\vec{y}'=a^{(L)}$. 设函数 h 是从输入到输出的前向传播，记为

$$h_\theta(\vec{x}) = \vec{y'} \qquad (3)$$

Algorithm 1: The *back-propagation* algorithm for performing batch-training of an ANN.

procedure: $\texttt{train}_{GD}(\theta, X, Y, max\_iter)$

$\theta \leftarrow \mathcal{U}(-\frac{1}{\|l^{(1)}\|}, \frac{1}{\|l^{(1)}\|})$. ▷ random initialization

cur_iter $\leftarrow 0$

while *cur_iter* $\leq$ *max_iter* **do**

  $A, Y' \leftarrow h_\theta(X)$ ▷ *forward propagation*

  $deltas \leftarrow b_\theta(Y, Y')$ ▷ *backward propagation*

  $\theta \leftarrow \texttt{GD}_\ell(A, deltas)$ ▷ *weight update*

  cur_iter++

end

return θ

C. 训练人工神经网络

人工神经网络的输出取决于训练集和训练算法。一个训练数据集由实例$\vec{x} \in$ X 和相应的期望输出$\vec{y} \in$ Y(例如，分类标签)组成。在训练过程中，神经网络的权重被调整为 $h_\theta(\vec{x}) = \vec{y}$ 常用的训练神经网络的算法(即，在给定 X，Y 的情况下找到最佳的 W 和$\vec{b}$)被称为反传算法[6]。

反传算法包括期望 y'和期望 y 之间的误差通过输出传播到每个神经元。在这个过程中，神经元的实际激活和期望激活之间的误差被存储。我们将这种反向传播过程表示为函数 b_θ(Y，Y')。给定 h_θ 执行量，让 A 是每个神经元的激活值，让 Delta 是所有神经元的激活误差值。

给定当前的 A 和 Delta，以及给一个设定的学习速率 $l \in (0,1]$，我们可以通过执行梯度下降(GD)算法[26]来逐步优化 W 和 b。综上所述，反向传播算法如下:

上述过程被称为批处理训练。这是因为对于每次迭代，GD 根据 x 中所有实例的集体误差更新权重。另一种方法是随机训练，其中使用随机梯度去气味(SGD)来代替 GD。在 SGD 中，根据每个实例的误差分别更新权重。使用 SGD，那反向传播算法变为：

GD 和 SGD 的区别在于 GD 比 SGD 更好地收敛于最优值，但是 SGD 的初始收敛比较快[27]。关于反向传播算法的更详细解释，我们请读者参考[26]。

在 Kitsune，我们使用最大迭代次数为 1 的 SGD。换句话说，当我们处于训练阶段时，如果一个新的实例到达，我们将在算法 2 中执行内部循环的单次迭代，丢弃该实例，然后等待下一个实例。通过这种方法，我们只从每个观察到的实例中学习一次，并且保持在线算法。

Algorithm 2: The *back-propagation* algorithm for performing stochastic-training of an ANN.

procedure: $\texttt{train}_{SGD}(\theta, X, Y, max\_iter)$

$\theta \leftarrow \mathcal{U}(-\frac{1}{\|l^{(1)}\|}, \frac{1}{\|l^{(1)}\|})$. ▷ random initialization

cur_iter $\leftarrow 0$

while *cur_iter* $\leq$ *max_iter* **do**

  for x_i *in* $\|X\|$ **do**

    $A, y' \leftarrow h_\theta(x_i)$ ▷ *forward propagation*

    $deltas \leftarrow b_\theta(\vec{y}, \vec{y'})$ ▷ *backward propagation*

    $\theta \leftarrow \texttt{GD}_\ell(A, deltas)$ ▷ *weight update*

  end

  cur_iter++

end

return θ

D. 自动编码器

自动编码器是一种人工神经网络，它被训练来重建其输入(即 X＝Y)。具体来说，在训练期间，自动编码器试图学习该函数

$$h_\theta(\vec{x}) \approx \vec{x} \tag{4}$$

可以看出，自动编码器本质上是试图学习原始数据分布的恒等函数。因此，对网络施加约束，迫使它学习更有意义的概念和$\vec{x}$ 中特征之间的关系。最常见的约束是限制网络内层的神经元数量。狭窄的通道使网络学习输入的实例紧压缩编码和译码。

作为示例，图 2 说明了一个自动编码器，他在$1^{(1)}$层接收实例$\vec{x} \in \mathbb{R}^3$, (2) 在$1^{(2)}$层编码 (压缩)$\vec{x}$ ，和(3)在$1^{(3)}$层解码压缩的$\vec{x}$ 。如果自动编码器在层大小上是对称的，那么可以使用相同的(镜像的)权重来进行编码和解码[28]。这个技巧减少了训练中所需的计算量。例如，在图 2 中，$W^{(2)} = [W^{(1)}]^T$。

E. 用自动编码器进行异常检测

自动编码器已经被用于许多不同的机器学习任务。例如，生成新的内容[29]，并从图像中滤除噪声[30]。在本文中，我们对使用自动编码器进行异常检测很感兴趣。

一般来说，在 X 上训练的自动编码器有能力从与 X 相同的数据分布中重建看不见的实例。如果一个实例不属于从 X 中学到的概念，那么我们的期望重建会具有很大的误差。对于给定的自动编码器实例$\vec{x}$ 的重构误差，可以通过取$\vec{x}$ 和重构输出的$\vec{y}$ '之间的均方根误差(RMSE)来计算。两个向量之间的 RMSE 定义为

$$\mathrm{RMSE}\left(\vec{x},\vec{y}\right)=\sqrt{\frac{\sum_{i=1}^{n}\left(x_i-y_i\right)^2}{n}} \qquad (5)$$

其中 n 是输入向量的维数。

设 ϕ 为异常阈值，初始值为-1，设 β ∈ [1，∞)为某个给定的灵敏度参数。可以通过执行以下步骤将自动编码器应用于异常检测任务:

1) 训练阶段:在干净(正常)数据上训练自动编码器。对于训练集中的每个实例 x_i：

a) Execute: $s=\mathrm{RMSE}\left(\vec{x},h_\theta(\vec{x})\right)$
b) Update: if$(s\geq\phi)$ then $\phi\leftarrow s$
c) Train: Update θ by learning from x_i

2) 执行阶段: 当未知实例$\vec{x}$ 到达时:

a) Execute: $s=\mathrm{RMSE}\left(\vec{x},h_\theta(\vec{x})\right)$
b) Verdict: if$(s\geq\phi\beta)$ then *Alert*

Kitsune 对一组自动编码器执行异常检测的过程将在后面的 IV-D 部分详细介绍

F. 复杂度

为了激活 $l^{(i+1)}$层，必须执行(1)中所述的矩阵乘法 $W^{(i)}\cdot\vec{a}^{(i)}$。因此，激活 $l^{(i+1)}$层的复杂性是 $O(l^{(i)}\cdot l^{(i+1)})^2$。因此，执行人工神经网络的总复杂性取决于层数和每层中神经元的数量。使用 SDG(算法 2)在单个实例上训练人工神经网络的复杂度大约是执行复杂度的两倍。这是因为反向传播步骤。

我们注意到自动编码器可以很深的(有许多隐层)。一般来说，更深更广的网络可以学习更复杂的概念。然而，如上所示，深层网络的训练和执行在计算上可能很昂贵。这就是为什么在 KitNET 中，我们确保每个自动编码器限制于三层以内，最多有七个可见神经元。

四、Kitsune 网络入侵检测系统

在这一节中，我们将介绍 Kitsune NIDS: 数据包预处理框架、特征提取器和核心异常检测算法。我们还讨论了异常检测算法的复杂度，并对其运行时性能进行了限制。

A. 概述

Kitsune 是一个即插即用的网络入侵检测系统，它基于神经网络，旨在有效地检测网络流量中的异常模式。它通过(1)监控最近网络流量的统计模式，(2)通过一组自动编码器检测异常模式来运行。集成中的每个自动编码器负责检测与网络行为的特定方面相关的异常。由于 Kitsune 被设计成运行在简单的网络路由器上，并且是实时的，所以 Kitsune 被设计成内存占用小，计算复杂度低。

Kitsune 的框架由以下组件组成:

- **包捕获器：**负责查询原始数据包的外部库。示例库:NFQueue[31]、afpacket[32]和 tshark[33](Wireshark 的应用程序接口)。
- **包解析器：**负责解析原始数据包以获得特征提取器所需的元信息的外部库。示例库:Packet++[3]和 tshark。
- **特征提取器（FE）：**负责从到达的数据包中提取 n 个特征以创建实例$\vec{x} \in \mathbb{R}^n$。特征描述了数据包及其来源的网络信道。
- **特征映射器（FM）：**负责通过$\vec{x}$创建一组较小实例(表示为 v)并将 v 传递到异常检测器中。该组件还负责学习从$\vec{x} \rightarrow v$ 的映射。
- **异常检测器（AD）：**负责检测异常数据包，给定数据包的表示形式 v。

由于数据包捕获器和数据包提取器不是本文的贡献，我们将重点讨论 FE、FM 和 AD 组件。我们注意到 FM 和 AD 组件是任务通用的(即，仅取决于输入特征)，因此可以作为一个通用的在线异常检测算法重新应用。此外，我们还参考 AD 组件中的通用算法，称之为 KitNET。

KitNET 有一个主要输入参数 m: KitNet 集成中每个自动编码器的最大输入数。该参数影响 KitNET 中集成的复杂度。由于 m 涉及检测和运行时性能之间的权衡，因此 Kitsune 的用户必须决定哪个更重要(检测速率和包处理速率)。这种权衡将在第五节进一步讨论

FM 和 AD 有两种工作模式:训练模式和执行模式。对于这两个组件，经过用户定义的时间限制后，训练模式转换为执行模式。训练模式中的组件用给定的输入更新其内部变量，但不生成输出。相反，执行模式下的组件不会更新其变量，但会生成输出。

为了更好地理解 Kitsune 的工作原理，我们现在将描述数据包获取时的过程，如图 3 所示:

（1）包捕获器获取一个新数据包，并将原始二进制文件传递给包解析器。

（2）包解析器接收原始二进制文件并解析数据包，然后将数据包的元信息发送给 FE。例如，数据包的到达时间、大小和网络地址。

（3）FE 接收该信息，并使用它检索超过 100 个统计数据，这些统计数据用于隐式描述数据包来自的信道的当前状态。这些统计数据形成实例$\vec{x} \in \mathbb{R}^n$，并传递给 FM。

（4）特征映射器（FM）接收 $\vec{x}$...

- **训练模式：**...并使用 $\vec{x}$ 来学习特征图。该特征图将$\vec{x}$的特征分组分为每个最大大小为 m 的集合。在特征图分组完成前，不会向异常检测器（AD）传递任何信息。在训练模式结束时，特征图被传递给异常检测器（AD），异常检测器（AD）利用特征图来构建总体结构（每个集合构成集合中自动编码器的输入）。
- **执行模式：**...并学习完的特征图被用于从 $\vec{x}$ 创建小实例 v 的集合，之后该集合被传递给异常检测器（AD）的集成层中的各个自动编码器。

（5）异常检测器（AD）接收 v...

- **训练模式：**...如果并使用v训练集合层。正向传播的 RMSE 随后被用于训练输出层。输出层最大的 RMSE 被设置为 ϕ，并被存储以备后用。

- **执行模式：**...并跨所有层执行v。如果输出层的 RMSE 超过 ϕβ，则会记录一个带有数据包详细信息的警告。

（6）原始数据包，$\vec{x}$，和 v 被丢弃。

我们将更详细地讨论特征提取器（FM），特征映射器（FM）和异常检测器（AD）是如何工作的。

B. 特征提取器（FM）

特征提取是获取或设计描述真实世界观测值向量的过程。在网络异常检测中，提取特征是很重要的，这些特征可以捕获通过网络的每个数据包的上下文和目的。例如，考虑一个单一的 TCP 同步数据包。该数据包可能是与服务器建立连接的良性尝试，也可能是数百万个试图引起拒绝服务攻击（DoS）的类似数据包之一。再举一个例子，考虑从 IP 监控摄像机发送的视频流。尽管数据包的内容是合法的，但抖动可能会突然出现持续显著上升。这可能表明在中间人攻击中有人在嗅探流量。

这只是一些攻击的例子，时间统计特征可以帮助检测异常。从网络流量中提取这些类型的特征的挑战是(1)来自不同信道(会话)的分组被交错，(2)在任何给定时刻可能有许多信道，(3)分组到达率可能非常高。缺乏经验的方法是维护来自每个通道的数据包窗口，并持续计算这些窗口的统计数据。然而很明显，如何在内存方面变得不切实际，并且扩展性不是很好。

为此，我们设计了一个在动态数据流(网络信道)上快速提取时间统计特征的框架。该框架内存占用很小，因为它使用阻尼窗口上维护的增量统计数据。使用阻尼窗口意味着提取的特征是暂时的(捕获数据包通道的最新行为)，并且当其阻尼权重变为零时可以删除增量统计(节省额外的内存)。该框架具有 O(1)的复杂性，因为增量统计数据的集合是在哈希表中维护的。该框架还维护有用的 2D 统计数据，这些数据捕获连接的 tx 和 rx 流量之间的关系。

我们现在将简要描述阻尼增量统计是如何工作的。然后，我们将列举由特征提取器（FE）提取的统计特征以产生实例 $\vec{x}$。

（1）阻尼增量统计：$S = \{x_1, x_2, \ldots\}$是一个无界数据流，其中$x_i \in R$。例如 S 可以是一系列观察到的数据包大小。S 的均值、方差和标准差可以通过保持元组$IS \coloneqq (N, LS, SS)$，其中 N，LS 和 SS 是迄今为止看到的实例的数量、线性和平方和。具体来说，将 x_i 插入IS(IS←$N + 1, LS + x_i, SS + x_i^2$))，且任何给定时间的统计数据为：$\mu_s = \frac{LS}{N}, \sigma_s^2 = \left|\frac{SS}{N} - \left(\frac{LS}{N}\right)^2\right|, \sigma_S = \sqrt{\sigma_S^2}$。

为了提取数据流的当前行为，我们必须忘记旧的实例。缺乏经验的方法是保持值的滑动窗口。然而，这种方法的内存和运行时复杂性为 0(n)，与增量统计的 0(1)相反。此外，滑动窗口方法不考虑窗口跨越的时间量。例如，最后 100 个实例可能是在过去一小时或几秒钟内到达的。

对此的解决方案是使用阻尼增量统计。在阻尼窗模型中，旧值的权重随时间呈指数下降。假设 d 是衰减函数，定义为

$$d_\lambda(t) = 2^{-\lambda t} \qquad (6)$$

其中$\lambda > 0$是衰减因子，t 是从流 Si 进行最后一次观测以来经过的时间。阻尼增量统计量的元组定义为$IS_{i,\lambda} \coloneqq (\omega, LS, SS, SR_{ij}, T_{last})$，其中 w 是当前权重，$T_{\text{last}}$是$IS_{\text{i}}$，λ 的最后更新的时间戳，$RS_{\text{ij}}$是流 i 和 j 之间的残差乘积之和（用于计算 2D 统计量）。为了在时间t_{cur}用x_{cur}更新IS_{λ}，执行算法 3。

表 I：可以从 Si 和 Sj 计算的增量统计的摘要。

Type	Statistic	Notation	Calculation
1D	Weight	w	w
	Mean	μ_{S_i}	LS/w
	Std.	σ_{S_i}	$\sqrt{\lvert SS/w - (LS/w)^2 \rvert}$
2D	Magnitude	$\lVert S_i, S_j \rVert$	$\sqrt{\mu_{S_i}^2 + \mu_{S_j}^2}$
	Radius	R_{S_i,S_j}	$\sqrt{\left(\sigma_{S_i}^2\right)^2 + \left(\sigma_{S_j}^2\right)^2}$
	Approx. Covariance	Cov_{S_i,S_j}	$\frac{SR_{ij}}{w_i + w_j}$
	Correlation Coefficient	P_{S_i,S_j}	$\frac{Cov_{S_i,S_j}}{\sigma_{S_i}\sigma_{S_j}}$

表 I 提供了可以从增量统计$IS_{i,\lambda}$计算的统计列表。我们将计算涉及一个和两个增量统计的统计称为 1D 统计和 2D 统计。

Algorithm 3: The algorithm for inserting a new value into a damped incremental statistic.

procedure: update$(IS_{i,\lambda}, x_{cur}, t_{cur}, r_j)$

1 $\gamma \leftarrow d_\lambda(t_{cur} - t_{last})$ ▷ Compute decay factor

2 $IS_{i,\lambda} \leftarrow (\gamma w, \gamma LS, \gamma SS, \gamma SR, T_{cur})$ ▷ Process decay

3 $IS_{i,\lambda} \leftarrow (w+1, LS+x_{cur}, SS+x_i^2, SR_{ij}+r_i r_j, T_{cur})$ ▷ Insert value

4 return $IS_{i,\lambda}$

（2）为 Kitsune 提取的特性：每当数据包到达时，我们都会提取传递给定数据包的主机和协议的行为快照。快照由 115 个流量统计组成，捕捉到一个小的时间窗口：(1)数据包的一般发送者，(2)数据包发送者和接收者之间的流量。

具体来说，统计数据总结了所有的流量...

- ...源自该数据包的源 MAC 和 IP 地址(表示为 SrcMAC-IP)。
- ...源自该数据包的源 IP(表示为 SrcIP)。
- ...在该数据包的源 IP 和目的 IP 之间发送(表示为 Channel)。
- ...在数据包的源和目的 TCP/UDP 套接字之间发送（表示为 Socket）。

从单个时间窗 λ 中可以提取总共 23 个特征(捕捉上述特征)(见表 II)。特征提取器从总共五个时间窗口中提取相同的特征集：过去的 100 毫秒、500 毫秒、1.5 秒、10 秒和 1 分钟(λ = 5、3、1、0.1、0.01)，因此总共有 115 个特征。

表 II:数据包到达时从每个时间窗口 λ 提取的统计数据(特征)。

TABLE II: The statistics (features) extracted from each time window λ when a packet arrives.

The packet's...	Statistics	Aggregated by	# Features	Description of the Statistics
...size	μ_i, σ_i	SrcMAC-IP, SrcIP, Channel, Socket	8	*Bandwidth of the outbound traffic*
...size	$\|S_i, S_j\|, R_{S_i,S_j}, Cov_{S_i,S_j}, P_{S_i,S_j}$	Channel, Socket	8	*Bandwidth of the outbound and inbound traffic together*
...count	w_i	SrcMAC-IP, SrcIP, Channel, Socket	4	*Packet rate of the outbound traffic*
...jitter	w_i, μ_i, σ_i	Channel	3	*Inter-packet delays of the outbound traffic*

我们注意到，并非每个数据包都适用于每种信道类型(例如，如果数据包不包含 TCP 或 UDP 数据报，则没有套接字)。在这些情况下，这些特征被归零。因此，有限元传递给调频的最终特征向量$\vec{x}$总是R^n的一个成员，其中 n = 115。特征提取器（FE）传递给特征映射器（FM）的最终特征向量$\vec{x}$总是R^n，其中 n=115。

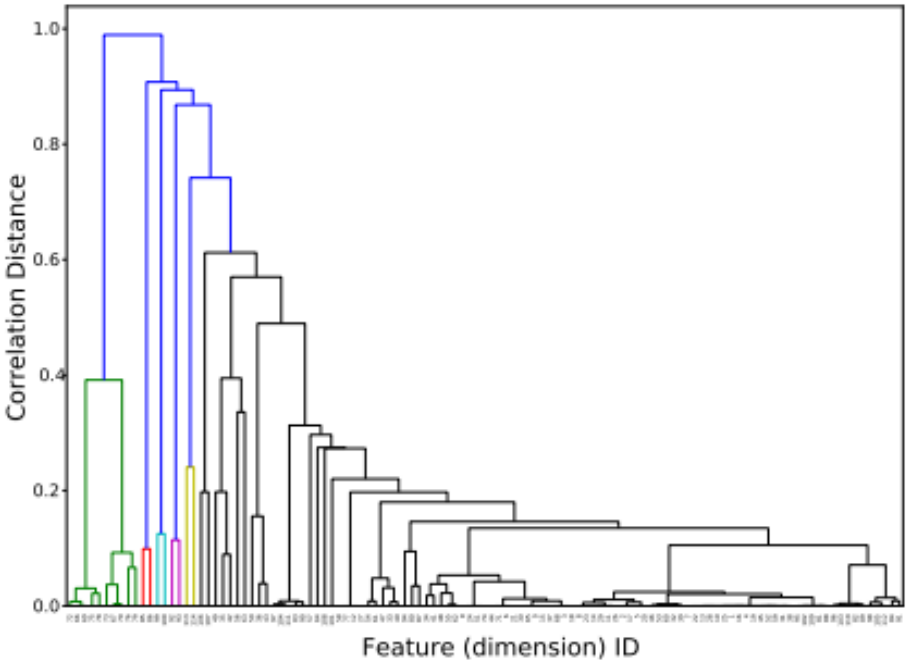


图 4:来自一百万个网络分组的聚集在一起的 115 个特征的示例树形图

C. 特征映射器（FM）

特征映射器（FM）的目的是将$\vec{x}$的 n 个特征（尺寸）映射到 k 个较小的子实例中，每个子实例对应于异常检测器（AD）集成层中的每个自动编码器。v 表示 k 个子实例的有序几个，其中：

$$v = \{\underset{v_1}{\rightarrow}, \underset{v_2}{\rightarrow}, \dots, \underset{v_k}{\rightarrow}\} \qquad (7)$$

我们注意到 v 的子实例可以看作$\vec{x}'s$的域 x 的子空间。

为了确保异常检测器（AD）中的系统以低复杂度有效运行，我们要求所选映射$f\left(\underset{x}{\rightarrow}\right) = v::$

（1）保证每个$\underset{v_i}{\rightarrow}$不超过 m 个特征，其中 m 是用户定义的系统参数。参数 m 影响集合的总体复杂性(见第Ⅳ-Ⅴ节)。

（2）将$\vec{x}$中的 n 个要素每一个准确地映射到 v 中的要素一次。这是为了确保集合不会太宽。

（3）包含 X 的子空间，这些子空间可以很好地捕获正常行为以检测相应子空间中发生的异常事件。

（4）在联机过程中被发现，以便一次不超过一个存储在内存中。

为了满足上述要求，我们通过将 X 的特征(维度)增量聚类到不大于 m 的 k 个组中来找到映射 f。我们通过对增量更新的汇总数据执行聚集层次聚类来实现这一点。

更准确地说，特征映射算法执行以下步骤：

（1）在训练模式下，使用实例$\vec{x}$的特性增量更新汇总统计信息。

（2）当训练模式结束时，对统计数据进行层次聚类，形成 f。

（3）在执行模式下，执行$\mathrm{f}\left(\underset{x_t}{\rightarrow}\right) = \mathrm{v}$，并将 v 传递给异常检测器（AD）。

为了确保分组特征捕捉正常行为，在聚类过程中，我们使用相关性作为二维之间的距离度量。通常，两个向量 u 和 v 之间的相关距离d_{cor}定义为

$$d_{cor}(u,v) = 1 - \frac{(u-\bar{u})\cdot(v-\bar{v})}{\|(u-\bar{u})\|_2\|(v-\bar{v})\|_2} \qquad (8)$$

其中 u 是向量$\vec{u}$中元素的平均值，u·v 是点积。

我们现在将解释为了聚类的目的，如何逐步概括特征之间的相关距离。令n_t为到目前为止看到的实例数。令$\vec{c}$为 n 维，其中包含每个要素的值的线性和，这样时间索引 t 处特征 i 的元素$c^{(i)} = \sum_{t=0}^{n_t} x_t^{(i)}$。类似地，$\underset{c_r}{\rightarrow}$表示包含每个特征的总和残差的向量，使得$c_r^{(i)} = \sum_{t=0}^{n_t}(x_t^{(i)} - \frac{c^{(i)}}{n_t})$。相似性，令$\underset{c_{rs}}{\rightarrow}$表示包含该特征的向量 每个特征的求和平方残差，使得$c_{rs}^{(i)} = \sum_{t=0}^{n_t}(x_t^{(i)} - \frac{c^{(i)}}{n_t})^2$。令 C 是 n×n 的部分相关矩阵，其中

$$[C_{i,j}] = \sum_{t=0}^{n_t}\left(\left(x_t^{(i)} - \frac{c^{(i)}}{n_t}\right)\left(x_t^{(j)} - \frac{c^{(j)}}{n_t}\right)\right) \qquad (9)$$

是特征 i 和 j 的残差之间的乘积之和。 设 D 是 X 的每个特征之间的相关距离矩阵。

$$D = [D_{i,j}] = 1 - \frac{C_{i,j}}{\sqrt{c_{rs}^{(i)}}\sqrt{c_{rs}^{(j)}}} \qquad (10)$$

现在我们知道了如何以增量方式获得距离矩阵，我们可以对距离矩阵进行聚集层次聚类来找到距离矩阵。简单地说，该算法从 n 个聚类开始，每个聚类对应一个由距离矩阵表示的点。然后，它搜索两个最近的点并加入它们相关的聚类。重复这个搜索和连接过程，直到有一个包含所有 n 个点的大簇。代表发现的链接的树被称为树形图(如图 4 所示)。有关聚类算法的更多信息，我们请读者参考[34]。通常，由于其复杂性，分层聚类不能在大型数据集上执行。然而，我们的距离矩阵很小(n 在几百个数量级)，因此现场计算是可行的。

最后，通过 D 的树状图，我们可以容易地找到没有聚类大于 m 的 k 个聚类(特征组)。该过程是断开树形图的最大链接(即最顶端的节点)，然后检查是否所有找到的聚类的大小都小

于 m。如果至少有一个聚类的大小大于 m，那么我们对超过的聚类重复该过程。在过程的最后，我们将有 k 组具有强互相关的特征，其中没有一个组大于 m。

本节中描述的算法适用于在线和现场处理，因为(1)它从不在内存中存储多个实例，(2)它使用非常少的内存(在列车模式下)，以及(3)非常快，因为更新过程需要更新小的 n 乘 n 距离矩阵。

D. 异常检测器(AD)

如图3所示，异常检测器（AD）组件包含一个特殊的神经网络，我们称之为KitNET (Kitsune NETwork)。KitNET 是一个无监督人工神经网络，设计用于在线异常检测任务。KitNET 由两层自动编码器组成:集成层和输出层。

- **集成层**：映射到 v 中各个实例的 k 个三层自动编码器的有序集合。该层负责测量 v 中每个子空间(实例)的独立异常。在训练模式期间，自动编码器学习它们各自子空间的正常行为。在训练模式和执行模式下，每个自动编码器向输出层报告其 RMSE 重建误差。
- **输出层**：一个三层自动编码器，学习集成层的正常(即训练模式)均方根值。该层负责产生最终异常分数，考虑(1)子空间异常之间的关系，以及(2)网络流量中自然出现的噪声。

我们现在将详细介绍 KitNET 如何操作集成层和输出层。

（1）初始化：当异常检测器（AD）从特征提取器（FM）接收第一组映射实例 v 时，异常检测器（AD）使用 v 作为蓝图初始化 KitNET 的体系结构。具体地，让 θ 表示整个自动编码器，并让$L^{(1)}$和$L^{(2)}$分别表示集合和输出层。$L^{(1)}$被定义为

$$L^{(1)} = \{\theta_1, \theta_2, \ldots, \theta_k\} \qquad (11)$$

有序集这样自动编码器$\theta_i \in L^{(1)}$有三层神经元：输入和输出中的$\dim\left(\underrightarrow{v_i}\right)$神经元，以及内层的$\left\lceil \beta \cdot \dim\left(\underrightarrow{v_i}\right)\right\rceil$神经元，其中β ∈ (0,1]（在我们的实验中，我们取 β= 3/4）图 5 说明了$\underrightarrow{v_i} \in v$和$\underrightarrow{\theta}_i \in L^{(1)}$之间的描述映射。

$L^{(2)}$被定义为单个自动编码器θ_0，其具有 k 个输入和输出神经元以及$\lceil k \cdot \beta \rceil$内部神经元。$L^{(1)}$和$L^{(2)}$层不通过公共 ANN 发现的加权突触连接。$L^{(2)}$的输入是来自$L^{(1)}$中每个相应自动编码器的 0-1 归一化的 RMSE 误差信号。信号$L^{(1)}$中每个自动编码器的聚合错误（RMSE），与来自$L^{(1)}$的每个神经元的信令相反，降低了网络的复杂性。最后，KitNET 中自动编码器θ_i的权重用随机值初始化 从均匀分布$\mathrm{u}(\frac{-1}{\dim\left(\underrightarrow{v_i}\right)}, \frac{1}{\dim\left(\underrightarrow{v_i}\right)})$。

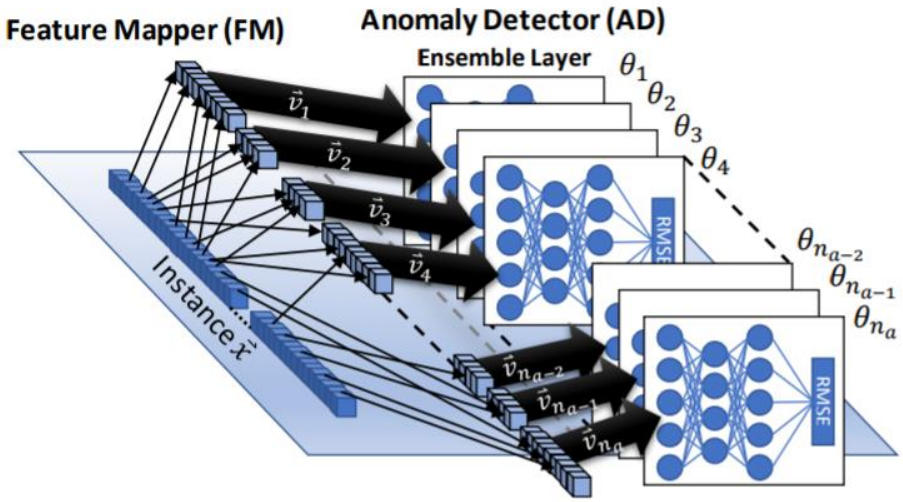


图 5:特征映射器和 KitNet 集成层间映射过程的图示：子实例$\underset{v_i}{\rightarrow}$从$\underset{x}{\rightarrow}$映射，然后发送到自动编码器$\theta_i$。

（2）训练模式：训练 KitNET 与训练常见的 ANN 网络略有不同，如第 III 部分所述。这是因为 KitNET 在网络的两个主要层之间发出 RMSE 重建错误信号。此外，KitNET 使用随机梯度下降（SGD）进行训练。 在一个实例上训练 KitNET 的算法在算法 4 中给出。

Algorithm 4: The *back-propagation* training algorithm for *KitNET*.

procedure: $\texttt{train}(L^{(1)}, L^{(2)}, \mathbf{v})$

// Train Ensemble Layer

1 $\vec{z} \leftarrow \text{zeros}(k)$ ▷ *init input for* $L^{(2)}$

2 **for** (θ_i *in* $L^{(1)}$) **do**

3 $\quad \vec{v_i'} = \texttt{norm}_{0-1}(\vec{v_i})$

4 $\quad A_i, \vec{y_i} \leftarrow h_{\theta_i}(\vec{v_i'})$ ▷ *forward propagation*

5 $\quad deltas_i \leftarrow b_{\theta_i}(\vec{v_i}', \vec{y_i})$ ▷ *backward propagation*

6 $\quad \theta_i \leftarrow \texttt{GD}_\ell(A_i, deltas_i)$ ▷ *weight update*

7 $\quad \vec{z}[i] \leftarrow \texttt{RMSE}(\vec{v_i}', \vec{y_i})$ ▷ *set error signal*

8 **end**

// Train Output Layer

9 $\vec{z}' = \texttt{norm}_{0-1}(\vec{z})$ $A_0, \vec{y_0} \leftarrow h_{\theta_0}(\vec{z}')$ ▷ *forward propagation*

10 $deltas_0 \leftarrow b_{\theta_0}(\vec{z}', \vec{y_0})$ ▷ *backward propagation*

11 $\theta_0 \leftarrow \texttt{GD}_\ell(A_0, deltas_0)$ ▷ *weight update*

12 return $L^{(1)}, L^{(2)}$

我们注意到，为了在第 3 行执行 0-1 归一化，每个自动编码器必须保存每个输入特征的最大值和最小值的记录。此外，这些最大和最小记录仅在训练模式期间更新。

与第III-E 节中的讨论类似，KitNET 必须接受正常数据的培训(不存在攻击)。这是[35]，[36]的一个常见假设，并且在许多类型的计算机网络中是实用的，例如知识产权摄像机监视系统。此外，还有过滤训练数据的方法，以便减少网络中可能存在的先前攻击的影响[37]，[38]。

Algorithm 5: The execution algorithm for *KitNET*.

procedure: execute($L^{(1)}, L^{(2)}, \mathbf{v}$)
// Execute Ensemble Layer
$\vec{z} \leftarrow \text{zeros}(k)$ ▷ *init input for* $L^{(2)}$
for (θ_i *in* $L^{(1)}$) **do**
  $\vec{v_i'} = \text{norm}_{0-1}(\vec{v_i})$
  $A_i, \vec{y_i} \leftarrow h_{\theta_i}(\vec{v_i'})$ ▷ *forward propagation*
  $\vec{z}[i] \leftarrow \text{RMSE}(\vec{v_i}', \vec{y_i})$ ▷ *set error signal*
end
// Execute Output Layer
$\vec{z}' = \text{norm}_{0-1}(\vec{z})$
$A_0, \vec{y_0} \leftarrow h_{\theta_0}(\vec{z_1}')$ ▷ *forward propagation*
return $\leftarrow \text{RMSE}(\vec{z}', \vec{y_0})$

（3）执行模式：在执行模式下，KitNET 不更新任何内部参数。相反，KitNET 在整个网络中执行正向传播，并返回$L^{(2)}$的 RMSE 重建误差。算法 5 给出了 KitNET 的执行过程。

$L^{(2)}$的重建误差测量实例关于 v 中子空间之间关系的异常。例如，考虑来自集合层$\theta_i, \theta_j \in L^{(1)}$的两个自动编码器。如果是在训练模式期间 RMSE$\theta_i$和$\theta_j$相关，那么执行模式中缺乏相关性可能被认为是比其独立 RMSE 的简单总和更为显着的异常（反之亦然）。由于输出层$L^{(2)}$学习在训练模式中这些关系（和其他复杂的关系），$L^{(1)}$的 RMSE 的$L^{(2)}$的重建误差将反映这些异常。

（4）异常评分：KitNET 的输出是 RMSE 异常评分$s \in [0, \infty)$，如第III节-E 所述。评分 s 越大，异常越大。要使用 s，必须确定异常得分截止阈值ϕ。最通常的方法是将ϕ设置为训练模式中看到的最大分数，我们假设所有情况都代表正常交通。另一种方法是概率选择ϕ。具体而言，可以(1)将输出的 RMSE 分数拟合为对数正态或非标准分布，然后(2)如果出现的概率非常低，则发出警报。KitNET 的用户应该根据他/她的算法应用来决定选择ϕ的最佳方法。在第Ⅴ节中，我们根据 Kitsune 的原始 RMSE 分数评估其探测能力。

E. 复杂性

作为基线，我们将把 KitNET 的复杂性与同一特征空间$\underset{x}{\rightarrow} \in R^n$上的单个三层自动编码器进行比较，压缩层比率为$\beta \in (0,1]$。

执行单个自动编码器的复杂性如下:在第III-第Ⅵ节中，我们发现激活层$l^{(i+1)}$在人工神经网络中的是$O(l^{(i)} \cdot l^{(i+1)})$。因此，执行单个自动编码器的复杂性是：

$$O(n \cdot \beta n + \beta n \cdot n) = O(n^2) \qquad (12)$$

执行 KitNET 的复杂性如下：我们提醒读者 k 表示特征映射器（FM）选择的子空间（自动编码器）的数量，m 是系统的输入参数，它定义了在$L^{(1)}$中任何给定自动编码器的最大输入数量。执行$L^{(1)}$和$L^{(2)}$的复杂度分别为$O(km^2)$和$O(k^2)$。由于变量$m \in 1,2,\ldots,10$是一个系统的常数参数，KitNET 的总复杂性是：

$$O(km^2 + k^2) = O(k^2) \qquad (13)$$

（13）的结果告诉我们，集合层的复杂度与 n 成线性比例，但输出层的复杂性取决于特征映射器（FM）指定了多少个自动编码器（子空间）。具体来说，最好的情况是$k = \frac{n}{m}$，在这种情况下性能增加了 m 倍。 最糟糕的情况是特征映射器（FM）在$L^{(1)}$中指定几乎每个单独的功能都有自己的自动编码器。在这种情况下，k = n，而 KitNET 作为单个宽自动编码器运行，这意味着没有性能增益。如果用户设置 m = 1 或 m = n 时会发生。

然而，后一种情况很少发生在自然数据集中。这是因为这将意味着来自第Ⅳ-C 特征映射器（FM）聚类过程的树木年谱是完全不平衡的。例如，

$$d_{cor}(x^{(1)}, x^{(2)}) < d_{cor}(x^{(2)}, x^{(3)}) < d_{cor}(x^{(3)}, x^{(4)}) < \ldots \qquad (14)$$

其中x^i是R^n的 i 维。因此，可以预期，在存在许多特性的情况下，KitNET 的运行时间将比单个自动编码器或堆叠式自动编码器更快。最后，训练 KitNET 的复杂性也是$O(k^2)$，因为我们只从每个实例中学习一次(见算法 4)。这可以与基于人工神经网络的分类器形成对比，该分类器通常在训练集上进行多次遍历(阶段)。

五、评价

在本节中，我们将从检测和运行时性能方面对 Kitsune 进行评估。我们首先描述数据集，然后是实验设置，最后是展示我们的结果。

A. 数据集

Kitsune 的目标是提供一个轻量级的入侵检测系统，它可以在一台简单的路由器上每秒处理许多数据包。鉴于这一目标，我们评估了 Kitsune 在真实的知识产权摄像机视频监控网络中检测攻击的能力。网络(如图 7 所示)由两个部署的四个高清监控摄像头组成。部署中的摄像机通过 PoE 供电，并通过点对点虚拟专用网隧道连接到数字录像机。远程站点的数字视频录像机通过客户端到站点的虚拟专用网络（VPN）连接为用户提供了对视频流的全局访问。网络中使用的摄像机及其配置如表Ⅳ所示。图 6 显示了我们设置中使用的八个摄像机中的两个。

在视频监控网络上可以进行多种攻击。但是，最关键的攻击会影响视频上行链路的可用性和完整性。例如，一个同步信号泛洪攻击了一个目标摄像机，或者一个人在攻击中涉及到

视频注入到实时视频流中。因此，在我们的评估中，我们将重点放在这些类型的攻击上。表III总结了我们实验中使用的攻击数据集，图 7 说明了每次攻击的攻击者位置(矢量)。表III中的违规列指出了攻击者对网络机密性(C)、完整性(Ⅰ)和可用性(A)的安全违规。如图 7 所示，从分组捕获点记录的所有数据集。

图 6：IP 摄像机视频监控网络中使用的两个摄像机。左侧:SNC-EM602RC。右侧:SNC-EB602R。

为了设置主动窃听，我们使用 Raspberry PI 3B 作为物理网桥。接口被提供了一个 USB 到以太网的适配器来提供第二个以太网端口，然后物理地放置在电缆的中间。

我们注意到，对于某些攻击，恶意数据包并没有明确地穿过 Kitsune 所连接的路由器。在这些情况下，由于网络行为的统计变化，特征提取器（FE）组件隐式捕获这些攻击。例如，中间人攻击影响数据包的定时，但不一定影响数据包本身的内容。

为了在 nosier 网络上评估 Kitsune，我们使用了一个额外的网络。额外的网络是一个无线网络，装有 9 个物联网设备和 3 台个人电脑。物联网是一个恒温器、婴儿监视器、网络摄像头、两个不同的门铃和四个不同的廉价安全摄像头。在这个特定的网络上，我们用 Mirai 未来组合僵尸网络恶意软件的真实样本感染了一个安全摄像头。在这个特定的网络上，我们用 Mirai 僵尸网络恶意软件的真实样本感染了一台安全摄像机。

表 3：用于评估 Kitsune 的数据集

Attack Type	Attack Name	Tool	Description: *The attacker…*	Violation	Vector	# Packets	Train [min.]	Execute [min.]
Recon.	**OS Scan**	Nmap	*…scans the network for hosts, and their operating systems, to reveal possible vulnerabilities.*	**C**	1	1,697,851	33.3	18.9
	Fuzzing	SFuzz	*…searches for vulnerabilities in the camera's web servers by sending random commands to their cgis.*	**C**	3	2,244,139	33.3	52.2
Man in the Middle	**Video Injection**	Video Jack	*…injects a recorded video clip into a live video stream.*	**C, I**	1	2,472,401	14.2	19.2
	ARP MitM	Ettercap	*…intercepts all LAN traffic via an ARP poisoning attack.*	**C**	1	2,504,267	8.05	20.1
	Active Wiretap	Raspberry PI 3B	*…intercepts all LAN traffic via active wiretap (network bridge) covertly installed on an exposed cable.*	**C**	2	4,554,925	20.8	74.8
Denial of Service	**SSDP Flood**	Saddam	*…overloads the DVR by causing cameras to spam the server with UPnP advertisements.*	**A**	1	4,077,266	14.4	26.4
	SYN DoS	Hping3	*…disables a camera's video stream by overloading its web server.*	**A**	1	2,771,276	18.7	34.1
	SSL Renegotiation	THC	*…disables a camera's video stream by sending many SSL renegotiation packets to the camera.*	**A**	1	6,084,492	10.7	54.9
Botnet Malware	**Mirai**	Telnet	*…infects IoT with the Mirai malware by exploiting default credentials, and then scans for new vulnerable victims network.*	**C, I**	X	764,137	52.0	66.9

表 4：实验中使用的摄像机的规格和数据

	SNC-EM602RC	SNC-EM600	SNC-EB600	SNC-EB602R
Resolution	1280x720			
Codec	H.264/MPEG4			
Frames/Sec	15			
Avg. Packets/Sec	195	350	290	320
Avg. Bandwidth	1.8 Mbit/s	1.4 Mbit/s	1.8 Mbit/s	1.8 Mbit/s
Protocol	RTP	RTP	Https *(TLSv1)*	Http/TCP

B. 实验设置

离线算法通常比在线算法执行得更好。这是因为离线算法在训练期间可以访问整个数据集，并可以对数据执行多次传递。然而，当资源(如计算能力和内存)有限时，在线算法是有用的。在我们的评估中，我们将 Kitsune 与在线和离线算法进行了比较。在线算法为 Kitsune 作为在线异常检测器提供了一个基线，而离线算法则为 Kitsune 作为一种上界的性能提供了一个视角。作为附加的基线，我们评估了 Suricata[39]——一个基于签名的 NIDS。Suricata 是一个开源的 NIDS，它类似于 Snort NIDS，但是在多个线程上并行化。与基于异常的 NIDS 相比，基于签名的 NIDS 的假阳性率要低得多。但是，它们无法检测未知的威胁或异常/抽象的行为。我们将 Suricata 配置为使用来自新兴威胁存储库[40]的 13465 条规则。

对于离线算法，我们使用了隔离森林(IF)[41]和高斯混合模型(GMM)[42]。IF 是一种基于集合的离群点检测方法，GMM 是一种基于期望最大化算法的统计方法。对于在线算法，我们使用了来自[43]和 pcStream2[44]的增量 GMM。pcStream2 是一种流聚类算法，它通过测量新实例到已知集群的 Mahalanobis 距离来检测离群值。

对于每个实验(数据集)，每个算法都在前一百万个数据包上训练，然后在剩余的数据包上执行。第一个百万包的持续时间取决于捕获时网络的包速率。例如，使用 OS Scan 数据集，每个算法在前 100 万个包上进行训练，然后在剩余的 697,851 个包上执行。表 3 列出了每个数据集的相关训练和执行周期。每个算法从表 2 获得完全相同的特性。

Kitsune 有一个主参数$m \in \{1,2,\dots,n\}$，这是 KitNET 集合中任意一个自动编码器的最大输入数。对于我们的检测性能评估，我们设置$m = 1$和$m = 10$。对于所有其他算法，我们使用默认设置。

C. 评价指标

异常检测器的输出是在$[0,\infty]$范围内的一个值，其中较大的值表示较大的异常(例如，自动编码器的 RMSE)。这个输出通常是标准化的，这样，值小于 1 的分数是正常的，大于 1 的分数是异常的。比分是规范化除以 cutt-off 阈值φ。选择φ对算法的性能有很大的影响。

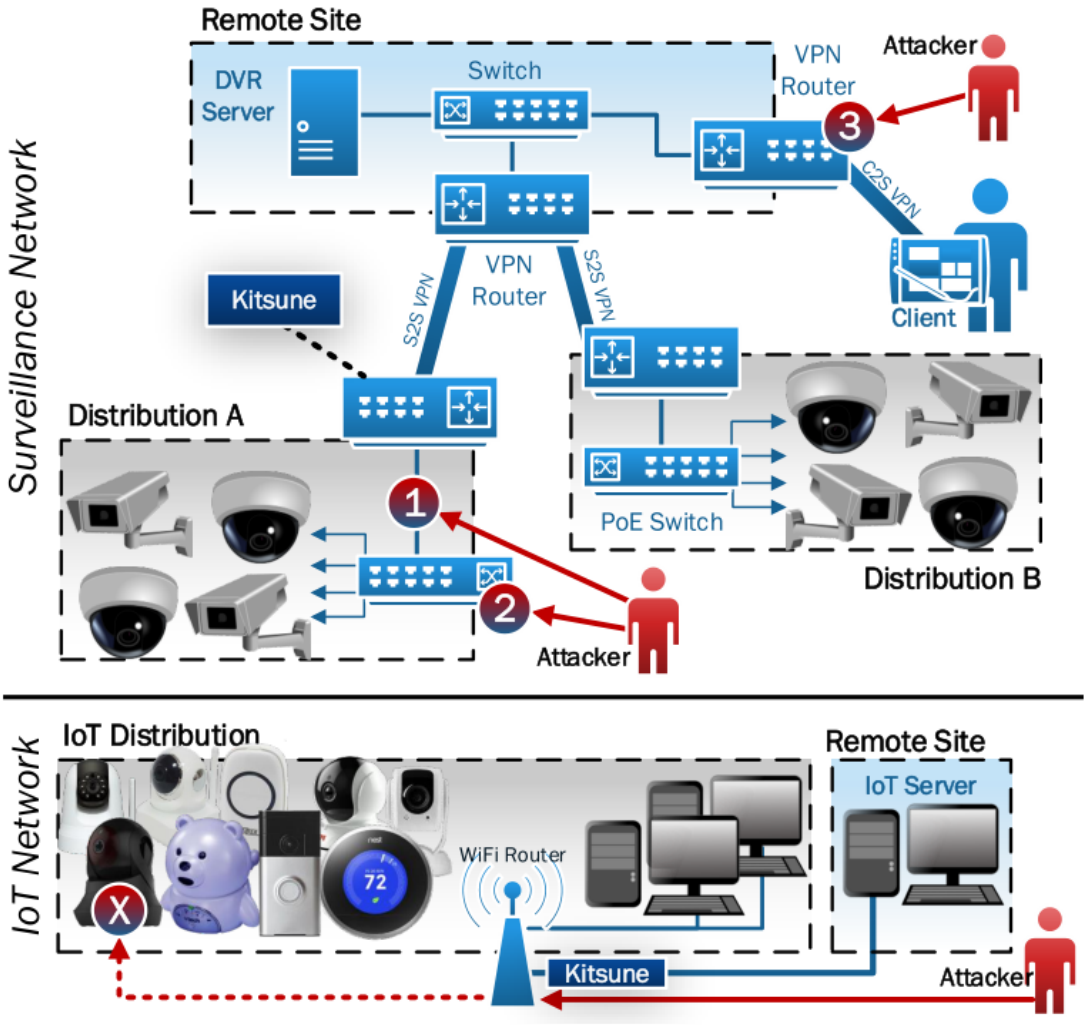


图 7:实验中使用的网络拓扑结构:监控网络(上)和物联网网络(下)

在特定数据集上，算法的检测性能可以通过其真阳性(TP)、真阴性(TN)、假阳性(FP)和假阴性(FN)来度量。在我们的评估中，当选择φ后，会计算出一个算法的真阳性率($TPR = \frac{TP}{TP+FN}$)和假阴性率($FNR = \frac{FN}{FN+TP}$)以保证假阳性率($FPR = \frac{FP}{FP+TN}$)很低(即,0.001)。我们还计算了 FPR 为 0 时的真阳性数。这些措施捕获了被正确检测到的恶意数据包的数量，并且误报或者遗漏的情况很少。在网络入侵检测中，因为分析人员需要花费大量的时间和金钱来调查每个警报，所以要最大程度上减少假警报的数量。

图 8 显示了在一次模糊攻击之前和期间，KitNET 的异常得分。较低的蓝线是我们在没有 FPs 产生的训练阶段可能选择的最低阈值。上面的蓝线是攻击过程中可能产生的最低阈值，它不会产生 FPs(类似全局最优)。查看这两个阈值的另一种方法类似于阈值选择的最佳情况和最差情况。因此，通过测量这两个阈值的性能，我们可以更好地了解算法作为异常检测器的潜力。

测量一般性能(即每一个可能的φ),我们使用了接受者操作特征曲线下面积(AUC)，以及平等的错误率(EER)。在我们的上下文中，AUC 是一个分类器将随机选择的异常实例的排名高于随机选择的正常实例的概率。换句话说，AUC 为 1 的算法是对给定数据集的完美异常检测

器，而 AUC 为 0.5 的算法是随机猜测标签。EER 是一种度量方法，它捕获了算法的 FNR 和 FPR 之间的权衡。当 FNR 和 FPR 最小且相等时，计算 FNR 和 FPR 的值。

D. 检测性能

图 9 为选取阈值时每个数据集上各算法的 TPR 和 FNR，使得 FPR 分别为 0 和 0.001。图中还显示了 AUC 和 EER。我们提醒读者，GMM 和 Isolation Forest 是批处理(离线)算法，它们可以完全访问整个数据集，并对数据集执行多次迭代。因此，这些算法是我们要达到的最优目标。在图 9 中，我们可以看到 Kitsune 相对于这些算法表现得非常好。特别是，Kitsune 在检测活动窃听方面的表现甚至比 GMM 更好。此外，我们的算法在 AR、Fuzzing、Mirai、SSL R.、SYN 和 active wiretap 数据集上的 EER 都优于 GMM。

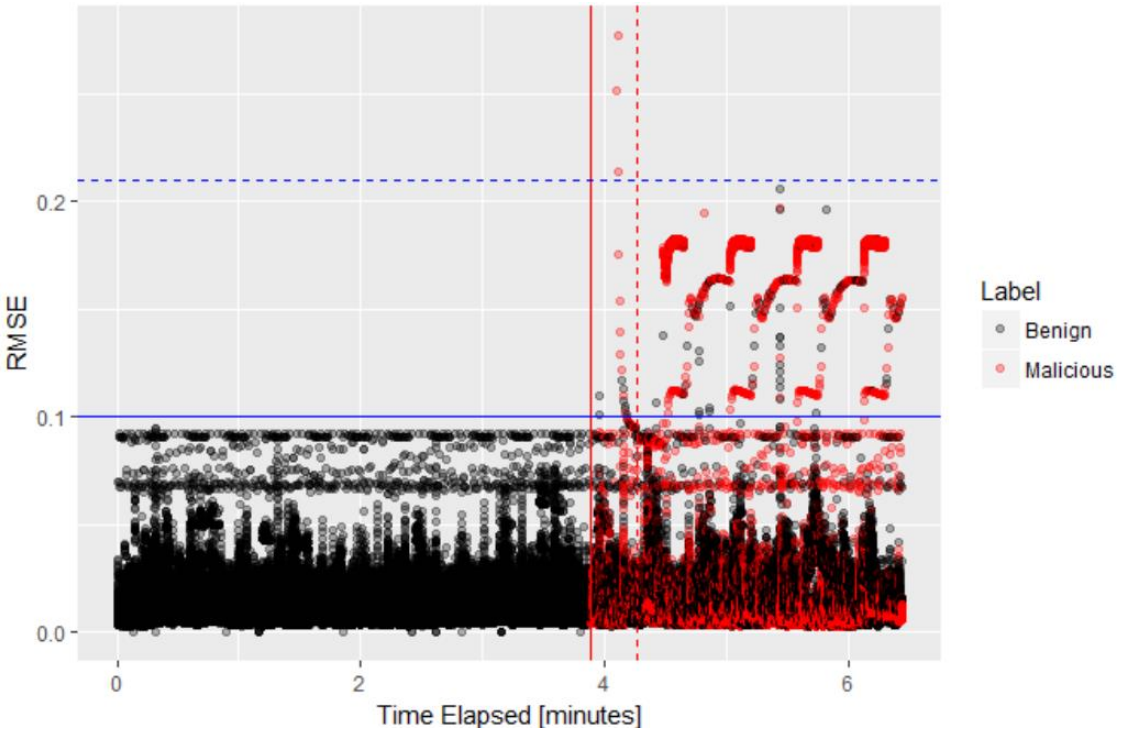


图 8:KitNET 的 RMSE 异常值在一次模糊攻击之前和期间。红线表示攻击者何时连接到网络(左)并发起攻击(右)

从图 9 可以看出，检测性能和 m(运行时性能)之间存在权衡。喜欢更好的检测性能而不是速度的用户应该使用接近 1 或 n 的 m, 而喜欢使用 speed 的用户则应该使用中等大小的 m。Kitsune 允许用户根据系统的要求来调整这个参数，m 对运行时性能的影响在 V-E 节中介绍。

作为在线算法性能的基准比较，我们将 Kitsune 与增量 GMM 和 pcStream2 进行了比较。总的来说，很明显，Kitsune 在 AUC 和 EER 方面同时胜过这两种算法。图 9 的第一行表示设置阈值使这里没有假阳性(如图 8 中的蓝色虚线条)时，每个算法能够获得的最大真阳性数。换句话说，这些图显示了每个异常检测程序是如何将恶意包的异常分数提高到噪声阈值之上的。这些数据显示，Kitsune 比其他算法更好地检测跨数据集的攻击，在大多数情况下比 GMM 更好。我们注意到$m = 10$的 Kitsune 有时比$m = 1$的表现更好。这是因为输出层自动编码器降低了集成的噪声层，这种影响会放大一些异常值的分数。

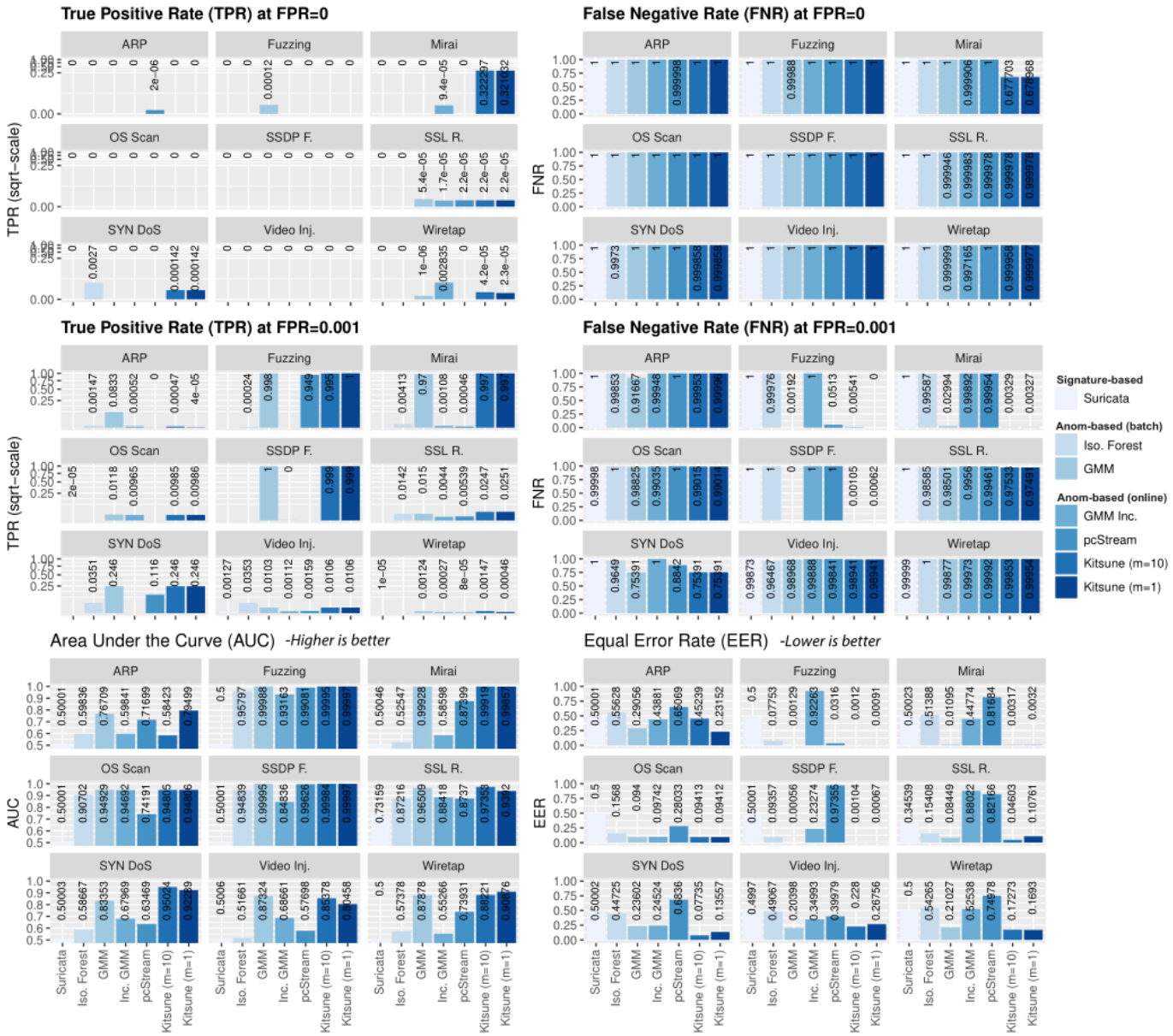


图 9:每个数据集上所有算法的实验结果:$FPR = 0.001$时的 TPR(左上)，$FPR = 0.001$时的 FNR(右上),AUC(左下),EER(右下)

E. 运行时性能

Kitsune 最大的优势之一是运行时性能。正如第 IV-E 节所讨论的，KitNET 的小型自动编码器集成比使用单个自动编码器更有效。这是因为集成减少了处理每个实例所需的操作总数。

为了演示这种效果，我们在 Raspberry PI 3B 和运行在 Windows 10 PC 上的 Ubuntu Linux VM 上进行了基准测试(完整的细节见表 V)。实验采用 C++语言编写，包含 n=198 个统计包特性，在单核(PI 上的物理核和 Ubuntu VM 上的逻辑核)上执行。

图 10 描绘了 KitNET 的集合大小对包处理速率的影响。使用$L^{(1)}$中的一个单独的自动编码器，PI 和 PC 每秒分别可以处理大约 1000 和 7500 个邮包。然而，在$L^{(1)}$中有 35 个自动编码器时，这两种环境的性能都提高了 5 到大约 5 倍。分别为 5400 和 37300。图 11 提供了 PI

在$k = 1$和$k = 35$时的包处理时间。该图显示，使用集成还可以减少处理时间中的差异。在不希望出现网络流量抖动的应用程序中，这可能是有益的。

树莓派的基准测试结果表明，一个简单的网络路由器，在有限的资源下，可以支持 Kitsune 作为一个 NIDS。这意味着 Kitsune 是一种廉价、可靠的分布式 NIDS 解决方案。我们注意到，由于实验是在单核机器上运行的，因此有可能进一步提高包处理的速率。为了实现这一点，我们计划在多个核上并行化执行 KitNET。

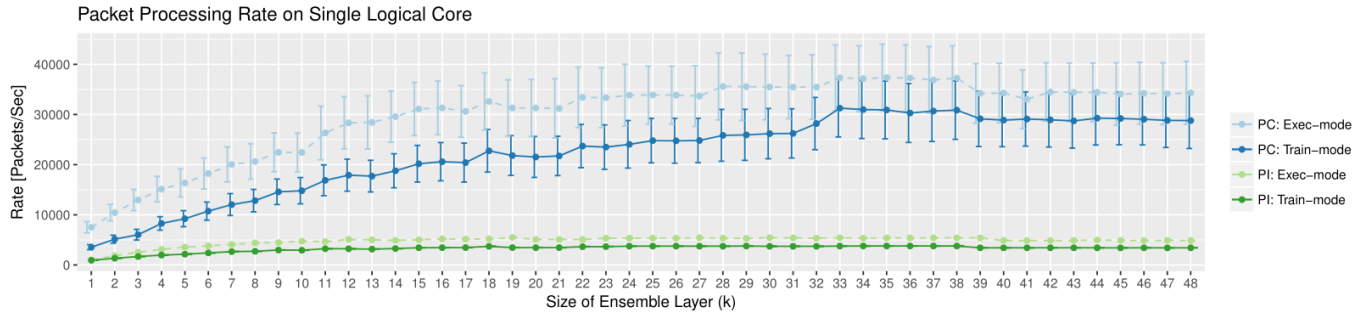


图 10:运行在 Raspberry PI 和 Ubuntu VM (PC)的单核上时，使用$n = 198$特征， KitNET 的总体大小(k)对平均包处理速率的影响

六、对抗性措施

使用 Kitsune 时，有几个方面需要考虑。首先，一个先进的对手可能尝试执行对抗性的机器学习[45]。当第一次安装时，在训练模式下 Kitsune 假设所有的通信都是良性的。因此，一个预先存在的对手可能能够逃避 Kitsune 的检测。然而，在执行模式下，Kitsune 会检测到新的攻击和新的威胁。无论如何，用户在安装 Kitsune 时应该意识到这种风险。在之后的工作中，如果能找到找到一种机制能够在训练过程中安全的过滤出可能被污染的实例，那将非常有趣。例如，我们可以首先执行并查看是否存在高异常值。如果有，那么我们将不会从实例中进行培训(因为我们只想从良性的实例中学习)。通过这样做，我们可以无限地保持在训练模式中。

有一个重要的问题是如果目标网络已经被污染，那么人们可能更喜欢使用基于签名的 NIDS，比如 Snort。需要权衡的是，基于签名的 NIDS 不能自动检测新的或抽象的威胁(如评估结果所示)。一个比较好的折衷方案是在 Kitsune 旁边安装一个高效的 NIDS(比如 Snort 3.0 或 Suricata)。

表 V:用于执行基准测试的环境

		Environment 1 **Raspberry PI 3B**	*Environment 2* **Ubuntu VM**
CPU	Type	Broadcom BCM2837	Intel i7-4790
	Clock	1.2GHz	3.60GHz
	Cores	4	4 (8 logical)
RAM		1 GB	4 GB

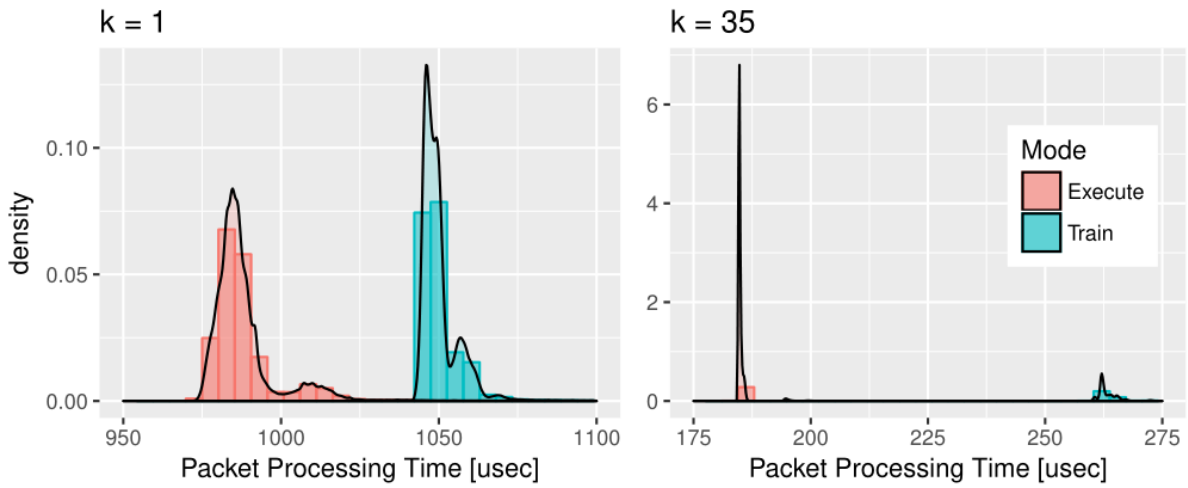


图 11:PI 中分组处理时间的密度图，$k = 1$(上)，$k = 35$(下)

Kitsune 面临的另一个威胁是针对 FE 的 DoS 攻击。在这种情况下，攻击者发送许多带有随机 IP 地址的数据包。通过这样做，FE 将创建许多增量统计信息，这些统计信息最终将消耗设备的内存。尽管这种攻击会导致明显的异常，但是系统可能会变得不稳定。因此，强烈建议用户限制可存储在内存中的增量统计信息的数量。可以注意到，使用 C++实现的 Kitsune，大约 1MB 的 RAM 可以包含大约 1000 个网络链接(假设每个链接有 5 个缓冲窗口)。一个很好的保持小内存占用的解决方案是使用$wi \approx 0$，周期性地搜索和删除增量统计数据。在实践中，因为我们使用相对较大的λs(快速衰变)，所以大多数的增量数据会保持在这种状态下。

七、结论

Kitsune 是一个基于神经网络的、高效并且即插即用 NIDS。它通过有效地跟踪所有网络通道的行为，并利用自动编码器(Kit-NET)集成进行异常检测来完成这项任务。在本文中，我们详细讨论了该框架的在线机器学习过程，并从检测和运行时性能方面对其进行了评估。KitNET 是一种在线算法，它的性能几乎与其他批处理/离线算法一样好，在某些情况下甚至更好。此外，该算法的效率足以运行在一个单核的树莓派上，如果在一个更强大的 CPU 中将会具有更大的潜力。

总而言之，在简单的网络设备上部署智能 NIDS 有很大的好处，特别是当整个部署过程是即插即用的时候。我们希望 Kitsune 对专业人员和研究人员都是有帮助的，并希望 KitNET 算法能够激发人们对进一步开发基于在线神经网络的异常检测领域的兴趣。

致　谢

作者想要感谢美国 NEC 公司的 Masayuki Nakae，感谢他的反馈和帮助建立监控摄像头部署。作者还要感谢 Yael Mathov、Michael Bohadana 和 Yishai Wiesner 在创建 Mirai 数据集方面提供的帮助。

参考文献

[1] Marshall A Kuypers, Thomas Maillart, and Elisabeth Paté-Cornell. An empirical analysis of cyber security incidents at a large organization.Department of Management Science and Engineering, Stanford Univer-sity, School of Information, UC Berkeley, 30, 2016.

[2] Dimitrios Damopoulos, Sofia A Menesidou, Georgios Kambourakis, Maria Papadaki, Nathan Clarke, and Stefanos Gritzalis. Evaluation of anomaly-based ids for mobile devices using machine learning classifiers. Security and Communication Networks, 5(1):3–14, 2012.

[3] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. Expert Systems with Applications, 39(1):424–430, 2012.

[4] Supranamaya Ranjan. Machine learning based botnet detection us-ing real-time extracted traffic features, March 25 2014. US Patent 8,682,812.

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3):15, 2009.

[6] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. Neural network design. Martin Hagan, 2014.

[7] Nidhi Srivastav and Rama Krishna Challa. Novel intrusion detection system integrating layered framework with neural network. In Advance Computing Conference (IACC), 2013 IEEE 3rd International, pages 682–689. IEEE, 2013.

[8] Reyadh Shaker Naoum, Namh Abdula Abid, and Zainab Namh AlSultani. An enhanced resilient backpropagation artificial neural network for intrusion detection system. International Journal of Computer Science and Network Security (IJCSNS), 12(3):11, 2012.

[9] Chunlin Zhang, Ju Jiang, and Mohamed Kamel. Intrusion detection using hierarchical neural networks. Pattern Recognition Letters,26(6):779–791, 2005.

[10] Rebecca Petersen. Data mining for network intrusion detection: A comparison of data mining algorithms and an analysis of relevant features for detecting cyber-attacks, 2015.

[11] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In Advances in neural information processing systems, pages 2377–2385, 2015.

[12] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, André CPLF de Carvalho, and Jo˜ ao Gama. Data stream clustering: A survey. ACM Computing Surveys (CSUR), 46(1):13, 2013.

[13] Garcia-Teodoro et. al. Anomaly-based network intrusion detection:Techniques, systems and challenges. computers & security, 28(1):18–28, 2009.

[14] Harjinder Kaur, Gurpreet Singh, and Jaspreet Minhas. A review of machine learning based anomaly detection techniques. arXiv preprint arXiv:1307.7286, 2013.

[15] Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. Information Sciences, 177(18):3799–3821, 2007.

[16] Taeshik Shon, Yongdae Kim, Cheolwon Lee, and Jongsub Moon. A machine learning framework for network anomaly detection using svm and ga. In Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC, pages 176–183. IEEE, 2005.

[17] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2):1153–1176, 2016.
[18] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In RAID, volume 4, pages 203–222. Springer, 2004.
[19] Miao Xie, Jiankun Hu, Song Han, and Hsiao-Hwa Chen. Scalable hypergrid k-nn-based online anomaly detection in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems, 24(8):1661–1670, 2013.
[20] Srinivas Mukkamala, Andrew H Sung, and Ajith Abraham. Intrusion detection using an ensemble of intelligent paradigms. Journal of network and computer applications, 28(2):167–182, 2005.
[21] Xiangzeng Zhou, Lei Xie, Peng Zhang, and Yanning Zhang. An ensemble of deep neural networks for object tracking. In Image Processing (ICIP), 2014 IEEE International Conference on, pages 843– 847. IEEE, 2014.
[22] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. Autoencoder-based feature learning for cyber security applications. In Neural Networks (IJCNN), 2017 International Joint Conference on, pages 3854–3861. IEEE, 2017.
[23] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), pages 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.
[24] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoen- coders with nonlinear dimensionality reduction. In Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, page 4. ACM, 2014.
[25] T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagationfor classification. International Journal of Computer Theory and Engineering, 3(1):89, 2011.
[26] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. Optimization for machine learning. Mit Press, 2012.
[27] Léon Bottou. Stochastic gradient descent tricks. In Neural networks: Tricks of the trade, pages 421–436. Springer, 2012.
[28] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learn- ing machine: a new learning scheme of feedforward neural networks. In Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, volume 2, pages 985–990. IEEE, 2004.
[29] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In European Conference on Computer Vision, pages 835–851. Springer, 2016.
[30] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec):3371–3408, 2010.
[31] Yusuke Sugiyama and Kunio Goto. Design and implementation of a network emulator using virtual network stack. In 7th International Symposium on Operations Research and Its Applications (ISORA08), pages 351–358, 2008.

[32] Eric Leblond and Giuseppe Longo. Suricata idps and its interaction with linux kernel.
[33] Borja Merino. Instant Traffic Analysis with Tshark How-to. Packt Publishing Ltd, 2013.
[34] Fionn Murtagh and Pedro Contreras. Methods of hierarchical clustering. arXiv preprint arXiv:1105.0121, 2011.
[35] Wenke Lee, Salvatore J Stolfo, et al. Data mining approaches for intrusion detection. In USENIX Security Symposium, pages 79–93. San Antonio, TX, 1998.
[36] Mingrui Wu and Jieping Ye. A small sphere and large margin approach for novelty detection using training data with outliers. IEEE transactions on pattern analysis and machine intelligence, 31(11):2088–2092, 2009.
[37] Niladri Sett, Subhrendu Chattopadhyay, Sanasam Ranbir Singh, and Sukumar Nandi. A time aware method for predicting dull nodes and links in evolving networks for data cleaning. In Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on, pages 304–310. IEEE, 2016.
[38] Deepthy K Denatious and Anita John. Survey on data mining techniques to enhance intrusion detection. In Computer Communication and Informatics (ICCCI), 2012 International Conference on, pages 1–5. IEEE, 2012.
[39] Suricata — open source ids / ips / nsm engine. https://suricata-ids.org/, 11 2017. (Accessed on 11/14/2017).
[40] Index of /open/suricata/rules.https://rules.emergingthreats.net/open/ suricata/rules/, 11 2017. (Accessed on 11/14/2017).
[41] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conferenceon, pages 413–422. IEEE, 2008.
[42] Douglas Reynolds. Gaussian mixture models. Encyclopedia of biometrics, pages 827–832, 2015.
[43] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In Proceedings of the ACM/IEEEinternational conference on Human-robot interaction, pages 255–262.ACM, 2007.
[44] Yisroel Mirsky, Tal Halpern, Rishabh Upadhyay, Sivan Toledo, andYuval Elovici. Enhanced situation space mining for data streams. InProceedings of the Symposium on Applied Computing, pages 842–849.ACM, 2017.
[45] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence, pages 43–58. ACM, 2011.