本次介绍Pratyush Mishra发表在USENIX Security2020上的神经网络安全推理工作 DELPHI: A Cryptographic Inference Service for Neural Networks

0. Background & Motivation

如下图1所示，两方下的神经网络安全预测推理着眼于不泄露用户数据和服务器模型的前提下，对用户数据实现推理预测服务，其应用场景和基础知识在之前的博客中介绍过多次，在这里不作赘述。本文主要的工作也是在保证安全性的基础上，尽可能的减少通信、提升效率。本文和之前介绍的XONN的思想很相似，都是借鉴神经网络方面的发展，在保证模型性能（例如准确性）的前提下简化模型，从而使得模型计算对安全计算更加友好高效，以便达到目的。大致来说，本文主要借鉴的神经网络架构搜索方面的技术，将神经网络中最常用的ReLU函数替换为平方函数，从而减少网络中的比较操作，提升效率。

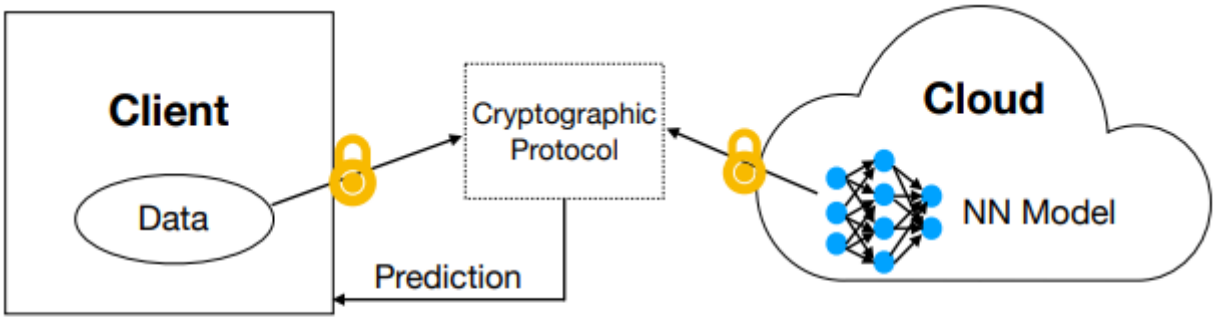


Figure 1: Cryptographic neural network inference. The lock indicates data provided in encrypted form.

本文用到了安全多方计算中的同态加密、秘密分享、混乱电路和茫然传输等知识。这些知识在之前的博文 ABY&ABY2.0 中都有过相关介绍。本文着眼于抵抗半诚实敌手。除此之外，本文还用到了神经网络架构搜索（NAS）。笔者对于NAS不是很熟悉，所以不作详细描述。而本文也大致是将NAS作为一种基本工具使用，重要工作还是集中在安全计算方面。

下面从安全协议和NAS对激活函数的优化两方面介绍DELPHI的工作。

1. Cryptographic Protocols

安全协议设计分为预处理和在线计算协议两部分，介绍如下。

1.1 Preprocessing Phase

在预处理阶段，服务器是有模型 $\mathbf{M}$， 但是用户的数据还没有输入系统。在这个阶段，用户和服务器首先生成随机数，这些随机数用户在线计算。

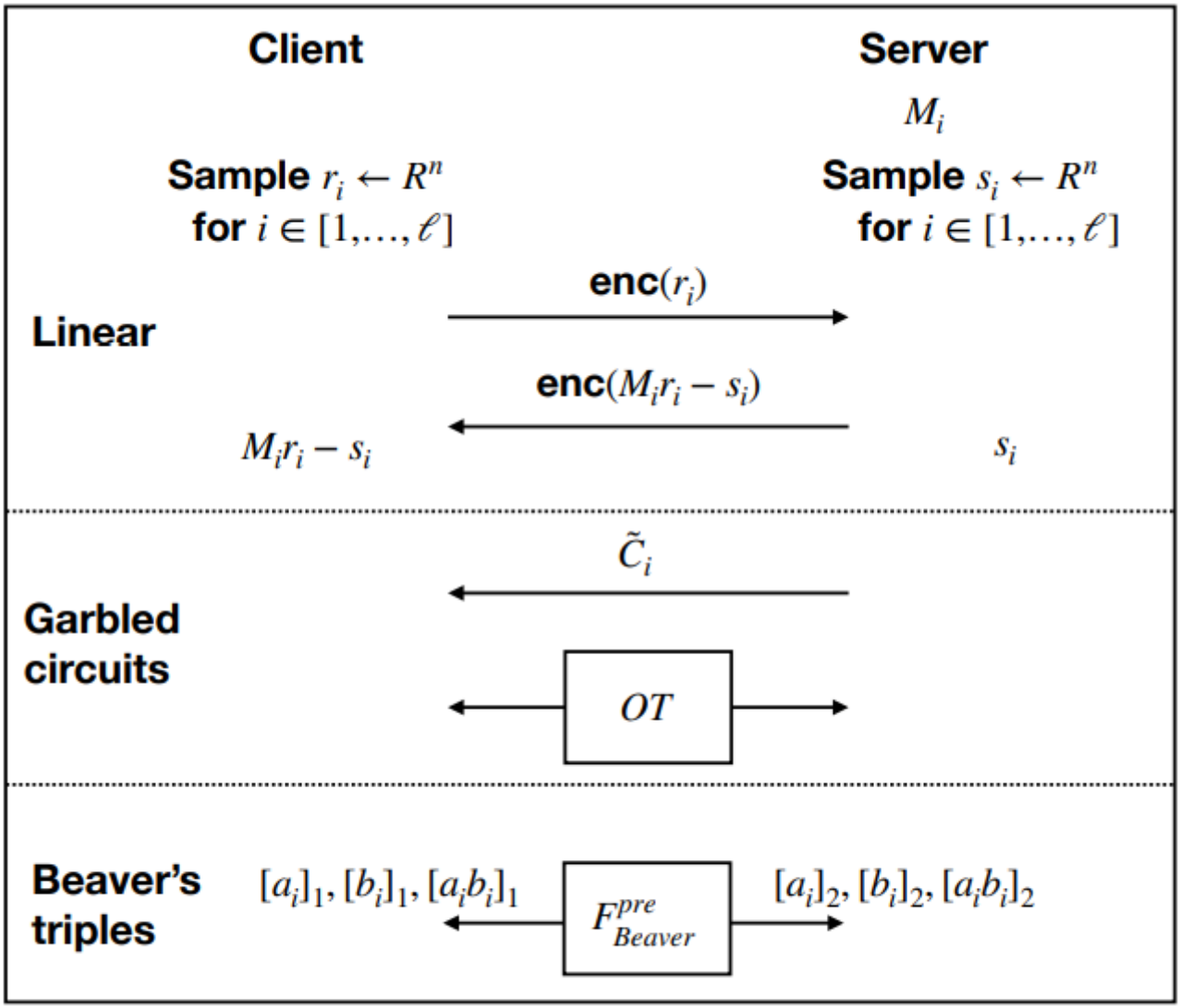


Figure 3: DELPHI's preprocessing phase.

对于线性层和激活层，预处理各有不同：

1. 对于第 i 个线性层 M_i，用户选择随机向量 $\mathbf{r}_i \leftarrow \mathbb{R}^n$， 同时服务器也生成随机向量 $\mathbf{s}_i \leftarrow \mathbb{R}^n$。随后，客户端发送同态加密密文 $\mathbf{enc}(\mathbf{r}_i)$ 给服务器，服务器在密文下计算 $\mathbf{enc}(M_i\mathbf{r}_i - \mathbf{s}_i)$，并将结果返回给客户端解密，得到 $M_i\mathbf{r}_i - \mathbf{s}_i$。而 $\mathbf{r}_i$ 将用于在线计算，盲化客户端真实数据；
2. 激活层处理分为ReLU和平方函数：
   - 针对ReLU函数，服务器生成对应的GC电路 $\widetilde{C}$，将 $\widetilde{C}$ 发送给客户端。同时，客户端和服务器执行OT协议，客户端作为接收方获取 $\widetilde{C}$ 中对应 $M_i\mathbf{r}_i - \mathbf{s}_i$ 和 $\mathbf{r}_{i+1}$ 的label；
   - 对于平方激活层，则需要生成Beaver三元组。生成方法利用同态或者OT都可以，具体可以参考 ABY & ABY2.0 一文。

1.2 Online Phase

在线阶段的计算流程如图4所示：

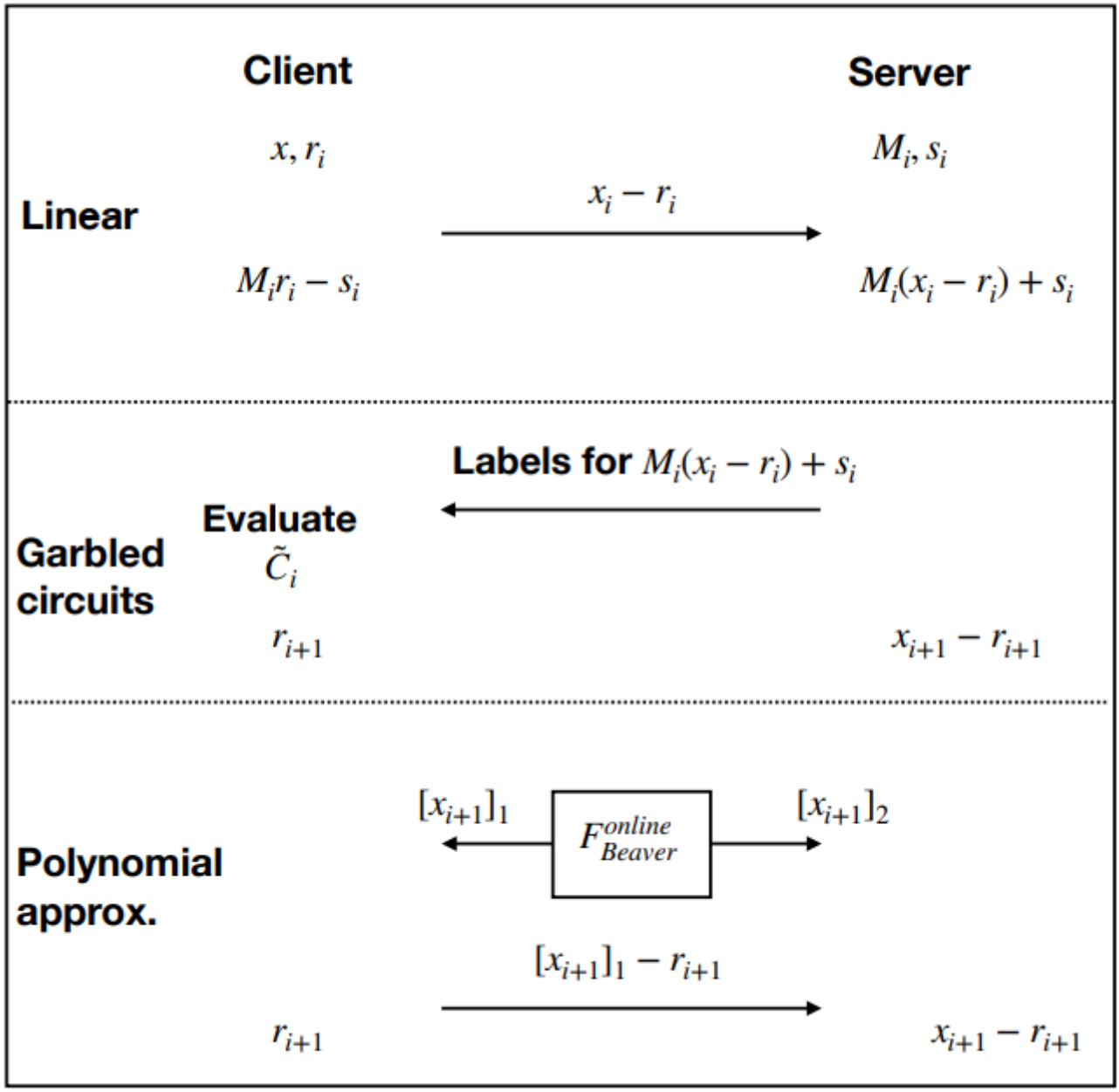


Figure 4: DELPHI's online phase.

首先客户端将输入 $\mathbf{x}$ 盲化，发送 $\mathbf{x} - \mathbf{r}_1$ 给服务器；接下来：

1. 对于第 i 个线性层，服务器计算 $M_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$，又因为在预处理阶段客户端得到了 $M_i\mathbf{r}_i - \mathbf{s}_i$， 因此双方具有了 $M_i\mathbf{x}_i$的加性秘密分享；
2. 对于激活层，则需要考虑ReLU 和平方激活层两种情况：
   - 如图5所示，对于ReLU函数，服务器将和 $M_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$ 对应的label发送给客户端，客户端计算电路 $\widetilde{C}$ 并得到OTP密文 $\mathrm{OTP}(\mathbf{x}_{i+1} = \mathbf{r}_{i+1})$，并将其发送给服务器；服务器利用OTP的密钥得到 $\mathbf{x}_{i+1} = \mathbf{r}_{i+1}$；
   - 对于平方激活函数，则利用Beaver三元组计算乘法，两方分别得到加法秘密分享 $[\mathbf{x}_{i+1}]_1$ 和 $[\mathbf{x}_{i+1}]_2$；客户端进一步将 $[\mathbf{x}_{i+1}]_1 - \mathbf{r}_{i+1}$ 发送给服务器，服务器在做加法获得 $\mathbf{x}_{i+1} = \mathbf{r}_{i+1}$。
3. 对于神经网络的输出层，服务器发送 $\mathbf{x}_\ell - \mathbf{r}_\ell$，客户端加上 $\mathbf{r}_\ell$ 得到处理结果 $\mathbf{x}_\ell$。

1.3 有限域内的定点算术运算

本文实现中使用了41比特的大素数 $p = 2061584302081$和11比特的定点表示。参数设置允许一次乘法不溢出。并且本文采用了SecureML中的截断方法。进一步，本文将定点计算嵌入到64比特浮点数的尾数部分。由于尾数部分可以表示 $[2^{-53}, 2^{53}]$，而实际的矩阵向量乘法结果是$\approx 2^{52}$ 的整数，因此可以容纳。故而，可以利用GPU加速线性层的代数计算。

2. Planner

本文利用NAS，在满足一定模型性能的前提下，尽可能使用平方函数为激活函数。Planner首先将ReLU替换为ReLU6，然后尝试将ReLU6替换为平方函数。其设置NAS的打分函数为

$$score(N) = acc(N)(1 + \frac{\#quad.activations}{\#total.activations})$$

实际实现中使用了Population-based Training (PBT) 算法。Planner的伪代码如下：

Planner $\begin{pmatrix} \text{all-ReLU6 neural network} & N \\ \text{training data} & D \\ \text{accuracy threshold} & t \end{pmatrix}$

1. Let the number of non-linear layers in N be n.
2. Initialize set of output networks $\mathcal{F}$.
3. For i in $\{n/2, \ldots, n\}$:
   (a) Compute the set of best performing models with i quadratic approximation layers: $\mathcal{S}_i \leftarrow \mathsf{PBT}(N, D, \mathsf{score}(\cdot))$.
   (b) Optimize hyperparameters for these models: $\mathcal{S}_i' \leftarrow \mathsf{PBT}(\mathcal{S}_i, D, \mathsf{score}(\cdot))$.
   (c) If for any $N_j \in \mathcal{S}_i'$ the accuracy of N_j is less than t, discard N_j.
   (d) Define N_i to be the network with the maximum score among the remaining networks.
   (e) Set $\mathcal{F} := \mathcal{F} \cup \{N_i\}$.
4. Output $\mathcal{F}$.

Figure 6: Pseudocode for DELPHI's planner.

个人水平有限，Planner部分就不多做介绍了。感兴趣的欢迎讨论。

3. Evaluations

- DELPHI使用SEAL实现同态加密，使用fancy-garbling实现GC部分。而且已经开源链接。并和GAZELLE比较了各项效率指标。
- 硬件条件：两台AWS instance，Intel Xeon 8000 CPU，3.0GHz, 16GB RAM, 4 threads each, NVIDIA Tesla V100 GPUs.
- 数据集--模型： CIFAR-10--7层CNN，CIFAR-100--ResNet32。

3.1 Microbenckmarks

conv. parameters				time (ms)		comm. (MB)	
input $C \times H \times W$	kernel $N \times K \times K$	stride & padding	system	preproc.	online	preproc.	online
$16 \times 32 \times 32$	$16 \times 3 \times 3$	(1, 1)	**DELPHI**	1255	**17.41**	10.48	**0.065**
			GAZELLE	—	1255	—	10.48
$32 \times 16 \times 16$	$32 \times 3 \times 3$	(1, 1)	**DELPHI**	1285	**17.17**	5.24	**0.020**
			GAZELLE	—	1285	—	5.24
$64 \times 8 \times 8$	$64 \times 3 \times 3$	(1, 1)	**DELPHI**	2775	**17.05**	5.24	**0.036**
			GAZELLE	—	2775	—	5.24

Table 1: Computation time and communication cost of ResNet-32 convolutions in DELPHI.

如表1所示，和GAZELLE相比，DELPHI卷积层提升在线时间 $80\times$，在线通信 $150\times$。虽然DELPHI预计算开销大，但是该开销不超过GAZELLE的在线计算开销。

conv. parameters			time (ms)		
input $C \times H \times W$	kernel $N \times K \times K$	stride & padding	$b = 1$	$b = 5$	$b = 10$
$16 \times 32 \times 32$	$16 \times 3 \times 3$	(1, 1)	0.34	1.07	2.75
$32 \times 16 \times 16$	$32 \times 3 \times 3$	(1, 1)	0.24	0.61	1.10
$64 \times 8 \times 8$	$64 \times 3 \times 3$	(1, 1)	0.24	0.37	0.616

Table 2: Computation time and communication cost of ResNet-32 convolutions in DELPHI when run on the GPU across different batch sizes b.

进一步，GPU可以进一步提升卷积性能。和表1中相比，同等卷积参数下，GPU能够提升性能$50-100\times$。而且，随着batch的增加，时间开销成亚线性增加。

activation function	time (μs) preproc.	online	comm. (kB) preproc.	online
Quad	9.6	0.04	0.152	0.008
ReLU	111.9	65.0	17.5	2.048

Table 3: Amortized computation time and communication cost of individual ReLU and quadratic activations in DELPHI.

对比ReLU和平方函数，平方函数显然更加高效。

3.2 Planner's Performance

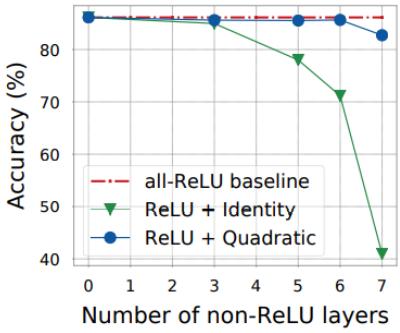


Figure 7: CIFAR-10 accuracy of 7-layer MiniONN networks found by our planner.

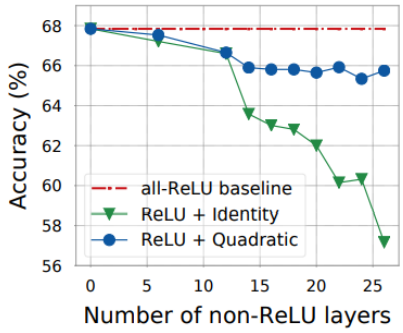


Figure 8: CIFAR-100 accuracy of ResNet-32 networks found by our planner.

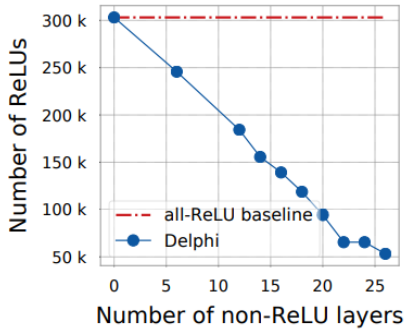
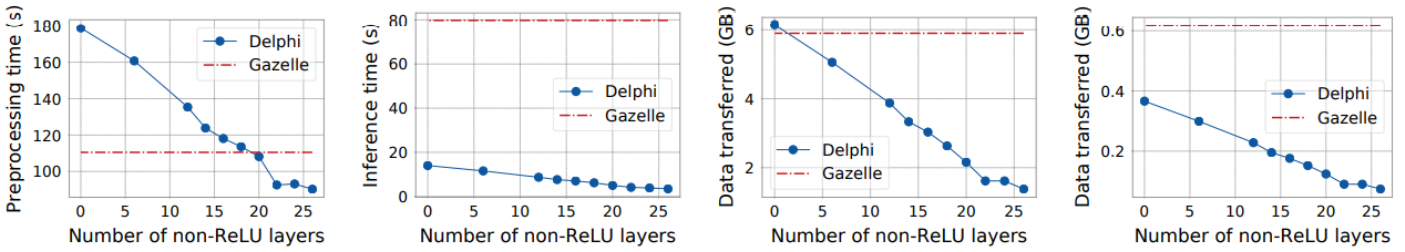


Figure 9: Number of ReLU activations in ResNet-32 networks found by our planner.

图7和8展示了Planner在满足一定条件下，准确率和Quad激活层数的关系。可以看出，使用Quad做激活函数仅仅带来了很少的准确率损失。比使用Identity函数好很多。另外，图9表示和GAZELLE中的ResNet32相比，本文利用Planner找到的网络具有更少的ReLU函数，因而更优。

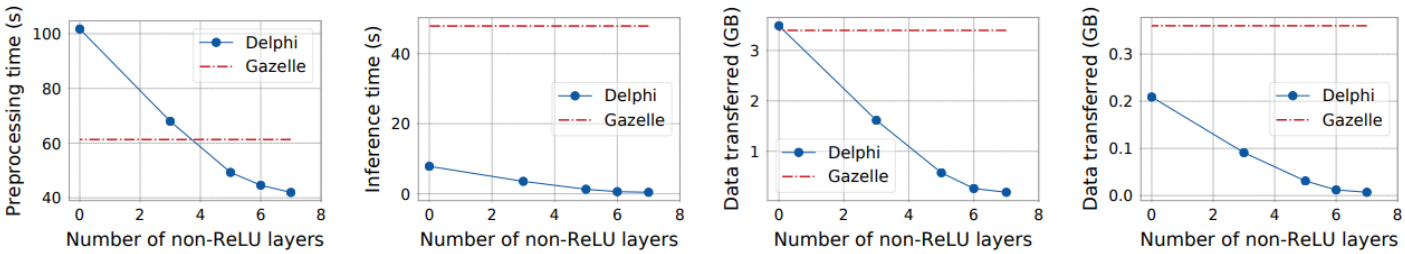
3.3 End-to-end Costs

图10和11展示了在DELPHI在最优网络上的运行时间和通信开销。GAZELLE运行在对应的all-ReLU网络上。可以看到，DELPHI大大降低了预计算和在线计算开销。



(a) Preprocessing time (b) Online time (c) Preprocessing communication (d) Online communication

Figure 12: Comparison of DELPHI with GAZELLE on the ResNet-32 architecture.



(a) Preprocessing time (b) Online time (c) Preprocessing communication (d) Online communication

Figure 13: Comparison of DELPHI with GAZELLE on the architecture from MiniONN [Liu+17a].

- 图12(a)和13(a)展示了预处理时间。对于没有NAS优化的网络，DELPHI预处理时间比GAZELLE更多，这是因为DELPHI需要更多时间对线性层进行预处理。而当深度优化之后，DELPHI的预处理时间则少于GAZELLE。同样的现象适用于12(c)和13(c)。对于planner找到的最优网络，DELPHI减少预处理时间 $1.5-2\times$，减少预处理通信 $6-40\times$。
- 对于在线开销，DELPHI提升时间效率 $22-100\times$，提升通信效率 $9-40\times$。

4. Conclusion

本文将NAS和安全计算结合起来，提出了更加高效的神经网络安全推理框架。并且，本文尝试用GPU加速线性层计算，算是比较早的该方面的工作。对CryptGPU也有一定的启发。
由于本人水平有限，对于NAS不是很了解，所以该方面只介绍了一点。非常欢迎对于该方面有兴趣的伙伴一起探讨、完善本篇博客。