ANEXO B: Passo-a-passo do Differential Privacy

O DP de exemplo se encontra no endereço a seguir:

https://github.com/rdmff/IN1098RobertoArthur/tree/main/DifferentialPrivacy

Após acessar o endereço, siga as etapas:

1. Busque um arquivo, com maior versão, nomeado:

DifferentialPrivacy_vX.Y.beta.ipynb

2. Abra no Jupyter Notebook do Anaconda ou Plataforma que carregue o formato que desejar.

3. Observe o bloco de imports e instale as dependências faltantes, inclusive Opacus:

pip install opacus==1.3.0

4. Execute os blocos de código (apenas os que têm código Python, ignorando os de texto/Markdown). Execute-os de cima para baixo, um após o outro.

5. O documento separa os testes em duas partes:

- **A) Baselines:** são as redes neurais, comuns, usadas – neste caso Convolutional Network. Observar gráficos e resultados.

- **B) DP:** parte onde o sistema irá adicionar ruído branco nos dados, escondendo informações pessoais dos dados originais. Vai gerar alguns gráficos, onde a precisão (*acuracy*) perdida é pouca, devido ao Opacus e ocultamento das informações do cliente. Observar gráficos e resultados.

Exercícios:

1) altere alguns parâmetros da seção de código “Ajustes de parâmetros” da parte B e execute novamente todos os passos, gerando novos gráficos e resultados. Altere: batch_size_dp para *32*, max_grad_norm para *1.5* e a privacidade para “*média*”.

2) Desafio. Dada a rede neural residual, abaixo, com padrões similares aos das anteriores, adicione esta e plote gráfico e resultados com as demais e o DP.

Arquivo, também, no repositório do GitHub com nome “**resnet.py**”:

“

```
# ResNet Block
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, 3, 1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, 1, stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        return F.relu(out)

class SimpleResNet(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(3, 64, 3, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(64, 2, stride=1)
        self.layer2 = self._make_layer(128, 2, stride=2)
        self.layer3 = self._make_layer(256, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(256, num_classes)

    def _make_layer(self, out_channels, blocks, stride):
        strides = [stride] + [1]*(blocks-1)
        layers = []
        for stride in strides:
            layers.append(ResidualBlock(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

“