## ANEXO A: Passo-a-passo do Federated Learning

Vide vídeo sobre FL no Readme.md do GitHub https://github.com/rdmff/IN1098RobertoArthur/ ou em https://drive.google.com/file/d/1eWy_pzRmufcCAdESJsww6VgZV9pBFvjU/ .

**Atividades:**

**Obs.: Lembre-se de devolver material com pesquisa, gráficos e comparações, similar aos código. Não apresente um trabalho com texto breve, escrito por extenso, mas um documento com código e simulação dos resultados!**

Tal qual no documento do GitHub federated_learning_atividade.ipynb .

```
# Imports necessários
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
import torchvision
import torchvision.transforms as transforms

import numpy as np
from collections import Counter, defaultdict

# Seed para reprodutibilidade
SEED = 42
torch.manual_seed(SEED)
np.random.seed(SEED)

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"🔧 Usando device: {device}")

# Configurações
```

```
NUM_CLIENTS = 3
NUM_CLASSES = 10  # CIFAR-10 tem 10 classes
UNDERSAMPLE_COUNT = 50  # Número de amostras para classes com undersample
NORMAL_SAMPLES_PER_CLASS = 800  # Número de amostras para classes normais

# Classes do CIFAR-10
CIFAR10_CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']

print(f"✅ Configurações definidas:")
print(f"   - Número de clientes: {NUM_CLIENTS}")
print(f"   - Número de classes: {NUM_CLASSES}")
print(f"   - Undersample count: {UNDERSAMPLE_COUNT}")
print(f"   - Normal samples per class: {NORMAL_SAMPLES_PER_CLASS}")
```

```
🩹 Usando device: cuda
✅ Configurações definidas:
  - Número de clientes: 3
  - Número de classes: 10
  - Undersample count: 50
  - Normal samples per class: 800
```

# 1. Carregamento de Dados

Carregamos o dataset CIFAR-10 com as transformações necessárias.

```
# Transformações para os dados
transform = transforms.Compose([
  transforms.ToTensor(),
  transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

# Carregar datasets
print("📥 Carregando CIFAR-10...")
trainset = torchvision.datasets.CIFAR10(
  root='./data',
  train=True,
  download=True,
  transform=transform
)

testset = torchvision.datasets.CIFAR10(
  root='./data',
  train=False,
  download=True,
  transform=transform
)

print(f"✅ Dataset de treino carregado: {len(trainset)} amostras")
print(f"✅ Dataset de teste carregado: {len(testset)} amostras")
print(f"✅ Classes: {CIFAR10_CLASSES}")
```

```
📥 Carregando CIFAR-10...
Files already downloaded and verified
```

```
Files already downloaded and verified
✅ Dataset de treino carregado: 50000 amostras
✅ Dataset de teste carregado: 10000 amostras
✅ Classes: ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
```

# 2. Separação de Clientes

Criamos 3 clientes, onde cada cliente possui dados com undersampling para 2 labels específicas.

```python
def create_non_iid_clients(dataset, num_clients, num_classes,
                           undersample_count=50, normal_samples=800):
    """
        Cria divisão não-IID dos dados onde cada cliente tem 2 classes com
undersample.

    Args:
        dataset: Dataset PyTorch
        num_clients: Número de clientes (3)
        num_classes: Número de classes (10 para CIFAR-10)
        undersample_count: Número de amostras para classes com undersample
        normal_samples: Número de amostras para classes normais

    Returns:
        Lista de datasets (um por cliente) e informações sobre distribuição
    """
    # Organizar índices por classe
    class_indices = defaultdict(list)
    for idx in range(len(dataset)):
        _, label = dataset[idx]
        class_indices[label].append(idx)

    # Embaralhar índices de cada classe
    for class_id in class_indices:
        np.random.shuffle(class_indices[class_id])

    client_datasets = []
    client_info = []
    used_indices = set()

    # Definir quais 2 classes terão undersample para cada cliente
    # Cliente 0: classes 0 e 1 com undersample
    # Cliente 1: classes 2 e 3 com undersample
    # Cliente 2: classes 4 e 5 com undersample
    undersample_classes_per_client = [
        [0, 1],  # Cliente 0
        [2, 3],  # Cliente 1
        [4, 5]   # Cliente 2
    ]
```

```
  # Para cada cliente
  for client_id in range(num_clients):
    client_indices = []
    client_dist = {}

    # Classes com undersample para este cliente
    undersample_classes = undersample_classes_per_client[client_id]

    # Para cada classe
    for class_id in range(num_classes):
      # Determinar quantas amostras pegar
      if class_id in undersample_classes:
        samples_to_take = undersample_count
      else:
        samples_to_take = normal_samples

      # Pegar amostras não usadas
          available = [idx for idx in class_indices[class_id] if idx not in
used_indices]
      selected = available[:samples_to_take]

      client_indices.extend(selected)
      used_indices.update(selected)
      client_dist[class_id] = len(selected)

    # Criar subset para o cliente
    client_dataset = Subset(dataset, client_indices)
    client_datasets.append(client_dataset)
    client_info.append({
      'distribution': client_dist,
      'undersample_classes': undersample_classes,
      'total_samples': len(client_indices)
    })

  return client_datasets, client_info

# Criar divisão dos clientes
print("🔄 Criando divisão de clientes...\n")
client_trainsets, client_info = create_non_iid_clients(
  trainset,
  NUM_CLIENTS,
  NUM_CLASSES,
  undersample_count=UNDERSAMPLE_COUNT,
  normal_samples=NORMAL_SAMPLES_PER_CLASS
)

# Imprimir informações de cada cliente
print("="*80)
print("📋 DISTRIBUIÇÃO DOS DADOS POR CLIENTE")
print("="*80)

for client_id, info in enumerate(client_info):
  print(f"\n👤 Cliente {client_id + 1}:")
```

```
  print(f"   Total de amostras: {info['total_samples']}")
            undersample_class_names    =    [CIFAR10_CLASSES[c]    for    c    in
info['undersample_classes']]
  print(f"   Classes com UNDERSAMPLE: {undersample_class_names}")
  print(f"   Distribuição por classe:")

  for class_id in range(NUM_CLASSES):
    count = info['distribution'][class_id]
     marker = "⚠ UNDERSAMPLE" if class_id in info['undersample_classes'] else
""
         print(f"         {CIFAR10_CLASSES[class_id]:12s}:  {count:4d}  amostras
{marker}")

print("\n" + "="*80)
🔄 Criando divisão de clientes...

================================================================================
=
📊 DISTRIBUIÇÃO DOS DADOS POR CLIENTE
================================================================================
=

📱 Cliente 1:
  Total de amostras: 6500
  Classes com UNDERSAMPLE: ['airplane', 'automobile']
  Distribuição por classe:
    airplane    :   50 amostras ⚠ UNDERSAMPLE
    automobile  :   50 amostras ⚠ UNDERSAMPLE
    bird        :  800 amostras
    cat         :  800 amostras
    deer        :  800 amostras
    dog         :  800 amostras
    frog        :  800 amostras
    horse       :  800 amostras
    ship        :  800 amostras
    truck       :  800 amostras

📱 Cliente 2:
  Total de amostras: 6500
  Classes com UNDERSAMPLE: ['bird', 'cat']
  Distribuição por classe:
    airplane    :  800 amostras
    automobile  :  800 amostras
    bird        :   50 amostras ⚠ UNDERSAMPLE
    cat         :   50 amostras ⚠ UNDERSAMPLE
    deer        :  800 amostras
    dog         :  800 amostras
    frog        :  800 amostras
    horse       :  800 amostras
    ship        :  800 amostras
    truck       :  800 amostras

📱 Cliente 3:
```

```
Total de amostras: 6500
Classes com UNDERSAMPLE: ['deer', 'dog']
Distribuição por classe:
  airplane    :  800 amostras
  automobile  :  800 amostras
  bird        :  800 amostras
  cat         :  800 amostras
  deer        :   50 amostras ⚠ UNDERSAMPLE
  dog         :   50 amostras ⚠ UNDERSAMPLE
  frog        :  800 amostras
  horse       :  800 amostras
  ship        :  800 amostras
  truck       :  800 amostras

================================================================================
=
```

# 3. Federated Learning

**Implemente aqui o código de Federated Learning usando o framework Flower.**

Os dados já estão preparados:

- `client_trainsets`: Lista com 3 datasets, um para cada cliente
- `testset`: Dataset de teste para avaliação
- `client_info`: Informações sobre a distribuição de dados de cada cliente

## Dicas:

- Use `flwr` (Flower) para implementar o federated learning
- Cada cliente deve treinar localmente com seu dataset (`client_trainsets[client_id]`)
- O servidor deve agregar os modelos dos clientes
- Use `testset` para avaliar o modelo global

```
# ================================================================================
# IMPLEMENTE AQUI O CÓDIGO DE FEDERATED LEARNING
# ================================================================================

print("📝 Implemente o código de Federated Learning acima!")
```

```
📝 Implemente o código de Federated Learning acima!
```