Análise de Vulnerabilidades em Gradient Inversion Attacks: Impacto da Variação Arquitetural e Aprimoramento de Técnicas de Ataque

[Cristiano Augusto Dias Mafuz]

23 de agosto de 2025

Resumo

Este trabalho investiga as vulnerabilidades de modelos de deep learning a ataques de inversão de gradientes (Gradient Inversion Attacks - GIA), com foco em duas linhas principais de pesquisa: (1) análise do impacto de diferentes arquiteturas de rede neural na eficácia dos ataques, e (2) desenvolvimento e avaliação de técnicas aprimoradas de reconstrução. Foram testadas múltiplas arquiteturas incluindo MLPs simples, redes profundas, CNNs e modelos com regularização (Dropout, BatchNorm). Além disso, foram propostos aprimoramentos nos algoritmos de ataque através da incorporação de termos de regularização adicionais (Total Variation, regularização espectral) e estratégias de otimização multi-candidato. Os resultados experimentais no dataset MNIST demonstram que (1) a arquitetura da rede tem impacto significativo na vulnerabilidade a GIA, com CNNs mostrando maior resistência que MLPs, e (2) as técnicas aprimoradas de ataque conseguem melhorar substancialmente a qualidade das reconstruções, especialmente quando combinadas com múltiplas estratégias de inicialização e regularização adaptativa.

1 Introdução

Os ataques de inversão de gradientes (Gradient Inversion Attacks) representam uma classe crítica de vulnerabilidades em sistemas de aprendizado federado e distribuído. Estes ataques exploram o fato de que gradientes compartilhados durante o treinamento podem inadvertidamente vazar informações sobre os dados privados utilizados [1].

O problema fundamental pode ser formalizado da seguinte forma: dado um modelo f_θ parametrizado por θ e um gradiente observado $\nabla_\theta \mathcal{L}(f_\theta(x), y)$, o objetivo é reconstruir os dados originais (x, y) que geraram esse gradiente. Este processo de reconstrução é tipicamente formulado como um problema de otimização:

$$\hat{x}, \hat{y} = \arg\min_{x', y'} \|\nabla_\theta \mathcal{L}(f_\theta(x'), y') - \nabla_\theta \mathcal{L}(f_\theta(x), y)\|^2 \tag{1}$$

1.1 Motivação e Objetivos

Embora trabalhos anteriores tenham estabelecido a viabilidade dos ataques GIA, questões fundamentais permanecem em aberto:

1. **Impacto Arquitetural**: Como diferentes arquiteturas de rede neural afetam a vulnerabilidade a ataques GIA? Redes mais profundas são mais ou menos vulneráveis? Como técnicas de regularização (Dropout, BatchNorm) influenciam a reconstrutibilidade?
2. **Aprimoramento de Ataques**: É possível desenvolver técnicas mais eficazes de reconstrução através de termos de regularização auxiliares e estratégias de otimização avançadas?

Este trabalho aborda essas questões através de uma análise experimental sistemática, contribuindo com:

- Análise comparativa de vulnerabilidades entre diferentes arquiteturas (MLPs, CNNs, redes profundas)

- Avaliação do impacto de técnicas de regularização na resistência a GIA
- Desenvolvimento de métodos aprimorados de reconstrução com múltiplos termos auxiliares
- Implementação de estratégias de otimização multi-candidato para melhor convergência

2 Trabalhos Relacionados

A área de ataques de inversão de gradientes (Gradient Inversion Attacks — GIA) tem evoluído rapidamente, com desenvolvimentos significativos tanto em técnicas de ataque quanto em mecanismos de defesa.

2.1 Trabalhos Fundamentais

Os trabalhos pioneiros estabeleceram a viabilidade de reconstruir dados privados a partir de gradientes. Zhu et al. [1] introduziram o ataque Deep Leakage from Gradients (DLG), formulando o problema como uma otimização que busca minimizar a distância entre os gradientes observados e os gerados por dados "dummy". Pouco depois, Geiping et al. [3] aprimoraram significativamente a estabilidade e eficácia do ataque com o Inverting Gradients (IG), introduzindo melhorias na inicialização e o uso de regularização de variação total. Paralelamente, Zhao et al. [2] propuseram o Improved DLG (iDLG), que melhorou a inferência de labels e a otimização, mostrando-se eficaz especialmente para reconstruir batches de tamanho um.

2.2 Evoluções Recentes e Técnicas Avançadas

Com base nos fundamentos, pesquisas subsequentes refinaram os ataques, tornando-os mais potentes e aplicáveis a cenários complexos. Um avanço notável foi a capacidade de reconstruir batches inteiros de imagens com alta fidelidade, como demonstrado por Yin et al. [5], que exploraram a direção dos gradientes para obter reconstruções mais precisas. Além disso, outros trabalhos estenderam a análise para diferentes tipos de dados, como grafos e dados espaço-temporais, e investigaram o uso de modelos generativos para aprimorar a qualidade das imagens recuperadas.

2.3 Mecanismos de Defesa

A comunidade de pesquisa respondeu com uma variedade de mecanismos de defesa. Uma abordagem comum é a Privacidade Diferencial, que adiciona ruído aos gradientes para ofuscar a informação privada, embora possa degradar a performance do modelo [8]. Outra linha de defesa foca em modificações na arquitetura do modelo para limitar o vazamento de informação. Trabalhos como Soteria [9] e Precode [10] propõem a inserção de módulos que restringem a informação contida nos gradientes, preservando a utilidade do modelo.

Simultaneamente, a compreensão dos modelos de ameaça foi aprofundada. Estudos como os de Wang et al. [4] e Fowl et al. [6] analisaram cenários de ataque mais realistas, investigando o vazamento de dados em nível de usuário e a possibilidade de ataques com acesso a modelos modificados, ampliando o panorama de vulnerabilidades em Aprendizado Federado.

2.4 Limitações dos Trabalhos Existentes

Apesar dos avanços, a literatura ainda apresenta lacunas. A maioria dos estudos foca em arquiteturas específicas, com pouca análise comparativa sobre como diferentes escolhas arquiteturais impactam a vulnerabilidade a GIA. Além disso, muitas avaliações ocorrem em cenários idealizados, e falta um consenso sobre métricas padronizadas para avaliar ataques e defesas de forma consistente.

Este trabalho se posiciona para endereçar essas limitações ao realizar uma análise sistemática do impacto arquitetural, propondo aprimoramentos técnicos em otimização e regularização e utilizando um conjunto abrangente de métricas para uma avaliação holística da eficácia dos ataques.

3 Fundamentação Teórica

3.1 Gradient Inversion Attacks

O ataque GIA clássico, introduzido por Zhu et al. [1], baseia-se na observação de que gradientes contêm informação sobre os dados que os geraram. O processo de reconstrução envolve:

1. **Observação**: Um adversário observa o gradiente $\nabla\mathcal{L}_{\text{real}} = \nabla_\theta\mathcal{L}(f_\theta(x_{\text{real}}), y_{\text{real}})$
2. **Inicialização**: Gerar estimativas iniciais $\hat{x}_0$ e $\hat{y}_0$
3. **Otimização**: Minimizar iterativamente a distância entre gradientes:

$$\mathcal{L}_{\text{grad}} = \|\nabla_\theta\mathcal{L}(f_\theta(\hat{x}), \hat{y}) - \nabla\mathcal{L}_{\text{real}}\|^2 \tag{2}$$

3.2 Termos de Regularização

Para melhorar a qualidade das reconstruções, são comumente utilizados termos auxiliares:

3.2.1 Total Variation (TV)

O termo de Total Variation promove suavidade espacial:

$$\mathcal{L}_{\text{TV}}(x) = \sum_{i,j} \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2} \tag{3}$$

3.2.2 Regularização L2

Penaliza magnitudes excessivas:

$$\mathcal{L}_{\text{L2}}(x) = \|x\|^2 \tag{4}$$

4 Metodologia

4.1 Seeds testadas e código inicial

Levando em consideração as capacidades do código disponibilizado originalmente, que executa um GIA clássico (sem técnicas complexas), foram usadas como teste seeds selecionadas de forma intencional, uma vez que, essas mesmas seeds aplicadas e reconstruídas pelo modelo original, retornava-se apenas uma imagem ruidosa. As seeds são: 47, 49, 51, 54, 68.

4.2 Arquiteturas Testadas

Já havia no código original, a seguinte "MLP Padrão"com 128 neurônios na camada oculta:

4.2.1 MLP Padrão

Listing 1: Implementação MLP Padrão

```
def create_simple_mlp(hidden_size=128):
    return nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, hidden_size),
        nn.ReLU(),
        nn.Linear(hidden_size, 10)
    )
```

Em adição, foram implementadas e avaliadas as seguintes arquiteturas:

4.2.2 MLP Raso

Listing 2: Implementação MLP Raso

```
def create_simple_mlp(hidden_size=64):
    return nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, hidden_size),
        nn.ReLU(),
        nn.Linear(hidden_size, 10)
    )
```

4.2.3 MLP Profundo

Listing 3: MLP com Múltiplas Camadas

```
def create_deep_mlp(hidden_sizes=[256, 128, 64]):
    layers = [nn.Flatten()]
    prev_size = 784
    for hidden_size in hidden_sizes:
        layers.extend([
            nn.Linear(prev_size, hidden_size),
            nn.ReLU()
        ])
        prev_size = hidden_size
    layers.append(nn.Linear(prev_size, 10))
    return nn.Sequential(*layers)
```

4.2.4 CNN

Listing 4: Rede Convolucional

```
def create_cnn_model():
    return nn.Sequential(
        nn.Conv2d(1, 32, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=3, padding=1),
        nn.ReLU(),
        nn.AdaptiveAvgPool2d((4, 4)),
        nn.Flatten(),
        nn.Linear(64 * 16, 128),
        nn.ReLU(),
        nn.Linear(128, 10)
    )
```

4.2.5 Com dropout

Listing 5: Rede com Dropout

```
def create_model_with_dropout(hidden_size=128, dropout_rate=0.3):
    return nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, hidden_size),
        nn.ReLU(),
        nn.Dropout(dropout_rate),
        nn.Linear(hidden_size, 10)
    )
```

4.2.6 Com Batchnorm

Listing 6: Rede com Batchnorm

```
def create_simple_mlp(hidden_size=128):
    return nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, hidden_size),
        nn.BatchNorm1d(hidden_size),
        nn.ReLU(),
        nn.Linear(hidden_size, 10)
    )
```

4.3 Aprimoramentos Propostos - Melhorias no Algoritmo de GIA

Esta seção detalha as melhorias implementadas no algoritmo básico de GIA, comparando a implementação original com as técnicas avançadas desenvolvidas.

4.3.1 Limitações da Implementação Original

A implementação base de GIA apresentava várias limitações que comprometiam a qualidade das reconstruções:

```
# Cidigo Original (reconstruct_single)
def reconstruct_single(model, g_star, lambda_reg=1e-4, iters=300, lr=0.1, seed=0,
    device='cpu'):
    torch.manual_seed(seed)
    x_hat = torch.rand(1, 1, 28, 28, requires_grad=True, device=device)
    y_logits = torch.zeros(1, 10, requires_grad=True, device=device)
    optimizer = optim.LBFGS([x_hat, y_logits], lr=lr, max_iter=20)

    def closure():
        optimizer.zero_grad()
        y_prob = y_logits.softmax(dim=-1)
        loss = (model(x_hat) * y_prob).sum()
        grads = torch.autograd.grad(loss, model.parameters(), create_graph=True)
        grad_diff = torch.cat([g.view(-1) for g in grads]) - g_star
        obj = grad_diff.pow(2).sum() + lambda_reg * x_hat.pow(2).sum() # Apenas L2
        obj.backward()
        return obj

    for _ in range(iters):
        optimizer.step(closure)
```

Limitações identificadas:

- Inicialização única: sem estratégia de múltiplos restarts;
- Regularização limitada: apenas termo L2 (lambda) básico;
- Otimização rígida: L-BFGS com configuração fixa;
- Ausência de early stopping: risco de overfitting;
- Falta de controle de gradientes: possibilidade de explosão de gradientes.

4.3.2 Função `Reconstruct_Advanced`: Melhorias Implementadas

Estratégia Multi-Restart

```
# Implementacao Aprimorada
for restart in range(restarts): # Multiplas tentativas
    # Inicializacao diferenciada por restart
    if normalize_data:
        x_hat = torch.randn(batch_size, 1, 28, 28, device=device) * 0.1
    else:
        x_hat = torch.rand(batch_size, 1, 28, 28, device=device) * 0.8 + 0.1
```

Justificativa: O problema de otimização não-convexo da GIA possui múltiplos mínimos locais. A estratégia multi-restart aumenta significativamente a probabilidade de encontrar soluções de melhor qualidade.

Regularização Multi-Termo

Total Variation (TV):

$$\mathcal{L}_{\text{TV}}(x) = \sum_{i,j} \sqrt{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2}$$

```
def total_variation_loss(x):
    return (torch.sum(torch.abs(x[:, :, :-1, :] - x[:, :, 1:, :])) +
            torch.sum(torch.abs(x[:, :, :, :-1] - x[:, :, :, 1:])))
```

Justificativa: promove suavidade espacial, atenuando ruídos de alta frequência sem distorcer significativamente as bordas principais, o que resulta em reconstruções visualmente mais limpas e coerentes.

Regularização Espectral:

$$\mathcal{L}_{\text{freq}}(x) = \alpha \sum_{u,v} |F(x)_{u,v}|^2 \cdot w(u,v)$$

onde $F(x)$ é a transformada de Fourier de x e $w(u,v)$ são pesos que penalizam altas frequências.

```
def frequency_regularization(x, alpha=1e-4):
    fft = torch.fft.fft2(x)
    h, w = x.shape[-2:]
    center_h, center_w = h // 2, w // 2
    y, x_coord = torch.meshgrid(torch.arange(h), torch.arange(w), indexing='ij')
    dist = torch.sqrt((y - center_h)**2 + (x_coord - center_w)**2)
    high_freq_mask = (dist > min(h, w) * 0.3).float().to(x.device)
    high_freq_penalty = torch.sum(torch.abs(fft) * high_freq_mask)
    return alpha * high_freq_penalty
```

Justificativa: imagens naturais seguem estatísticas específicas no domínio da frequência. A penalização de altas frequências promove reconstruções mais realistas.

TV Preservando Bordas:

$$\mathcal{L}_{\text{edge-TV}}(x) = \sum_{i,j} \frac{|\nabla x_{i,j}|}{1 + \beta |\nabla x_{i,j}|}$$

```
def edge_preserving_tv(x, beta=0.3):
    dx = x[:, :, :-1, :] - x[:, :, 1:, :]
    dy = x[:, :, :, :-1] - x[:, :, :, 1:]
    dx_abs = torch.abs(dx)
    dy_abs = torch.abs(dy)
    tv_x = torch.sum(dx_abs / (1 + beta * dx_abs))
    tv_y = torch.sum(dy_abs / (1 + beta * dy_abs))
    return tv_x + tv_y
```

Justificativa: parecido com o Total Variation em proposta: preserva bordas importantes, penalizando apenas ruído.

Ponderação Adaptativa:

```
# Pesos que evoluem durante a otimizacao
w_freq = min(1.0, iteration / 200)  # Aumenta com o tempo
w_edge = max(0.5, 1.0 - iteration / 800) # Diminui com o tempo

total_loss = (loss_grad + loss_l2 + loss_tv +
              w_freq * loss_freq + w_edge * loss_edge_tv)
```

Early Stopping Inteligente

```
# Sistema de early stopping com paciencia
if early_stopping:
    if current_loss < best_iter_loss:
        best_iter_loss = current_loss
        no_improve = 0
    else:
        no_improve += 1
        if no_improve > patience:
            break
```

Justificativa: diminui overfitting.

Projeção de Domínio

```
# Garantia de que reconstrucoes permanecem no dominio valido
with torch.no_grad():
    if normalize_data:
        x_hat.clamp_(-2.0, 2.0) # Para dados normalizados
    else:
        x_hat.clamp_(0.0, 1.0) # Para dados [0,1]
```

Gradient clipping

```
# [Otimizacao com gradient clipping]
# Gradient clipping
torch.nn.utils.clip_grad_norm_([x_hat, y_logits], max_norm=1.0)
```

Justificativa: prevenção de explosão de gradientes.

Otimização Adaptativa

```
# Selecao dinamica de otimizador e learning rate scheduling
optimizer = optim.Adam([x_hat, y_logits], lr=lr, betas=(0.9, 0.999))

def get_lr_schedule(optimizer, warmup_steps=100):
    def lr_lambda(step):
        if step < warmup_steps:
            return step / warmup_steps # Warmup
        else:
            return 0.95 ** ((step - warmup_steps) // 50) # Decay exponencial
    return optim.lr_scheduler.LambdaLR(optimizer, lr_lambda)
```

Melhorias:

- Adam: maior estabilidade e controle fino do learning rate;
- Learning Rate Scheduling: warmup inicial seguido de decay exponencial;

4.3.3 Algoritmo Multi-Candidato Completo

```
def multi_term_gia_attack(model, g_star, batch_size=1, device='cpu', **kwargs):
    best_result = None
    best_loss = float('inf')

    for restart in range(3): # Multiplos candidatos
        # [Inicializacao diferenciada por restart]

        for iteration in range(800):
            # [Computacao de multiplos termos de regularizacao]

            # Combinacao adaptativa de termos
            total_loss = (loss_grad + loss_l2 + loss_tv +
                          w_freq * loss_freq + w_edge * loss_edge_tv)

            # [Otimizacao com gradient clipping]

        # Selecao do melhor candidato
        if final_loss < best_loss:
            best_loss = final_loss
            best_result = (x_hat.detach().cpu(), predicted_labels, [final_loss], restart)

    return best_result
```

Justificativa: utiliza tanto TV Preservando Bordas, quanto Regularização Espectral, a depender do valor da Ponderação Adaptativa

4.3.4 Comparação Quantitativa das Melhorias

Aspecto	Implementação Original	Implementação Aprimorada
Restarts	1 (seed fixo)	1–5 (múltiplos)
Otimizador	L-BFGS fixo	Adam + LR scheduling
Regularização	Apenas L2	L2 + TV + Freq + Edge-TV
Early Stopping	Não	Sim (patience-based)
Gradient Control	Não	Clipping + normalization
Inicialização	Random uniforme	Adaptativa (norm/unnorm)
Iterações	300 fixo	500–1000 + early stop

4.3.5 Justificativa Teórica dos Aprimoramentos

Perspectiva Bayesiana: A formulação original pode ser vista como MAP estimation:

$$\hat{x} = \arg\max_x P(x|g) \propto P(g|x)P(x)$$

Onde $P(g|x)$ corresponde ao termo de matching de gradiente e $P(x)$ ao prior L2. Nossa abordagem expande o prior:

$$P(x) \propto \exp(-\lambda_{L2}\|x\|^2 - \lambda_{TV}\mathcal{L}_{TV}(x) - \lambda_{freq}\mathcal{L}_{freq}(x))$$

Análise de Convergência:

- Exploração inicial: LR alto com warmup;
- Refinamento: LR decrescente para convergência fina;
- Estabilidade: gradient clipping previne divergência.

Complexidade Computacional:

$$\text{Original: } O(T \cdot N) \qquad \text{Aprimorada: } O(R \cdot T \cdot N \cdot C)$$

Onde T = iterações, N = parâmetros, R = restarts, C = termos adicionais.
O overhead computacional (∼3–5x) é justificado pela melhoria significativa na qualidade.

4.3.6 Implementação de Referência

O algoritmo completo integra todos os aprimoramentos em uma função unificada `reconstruct_advanced()`, permitindo configuração flexível dos hiperparâmetros e ativação seletiva dos termos de regularização conforme a aplicação específica.

4.4 Métricas de Avaliação

Para avaliar a qualidade das reconstruções, utilizamos:

1. **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{5}$$

2. **Peak Signal-to-Noise Ratio (PSNR)**:

$$\text{PSNR} = 20\log_{10}\frac{\text{MAX}}{\sqrt{\text{MSE}}} \tag{6}$$

3. **Structural Similarity Index (SSIM)**:

$$\text{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{7}$$

4. **Acurácia de Labels**: Percentual de labels corretamente reconstruídos
5. **Loss**: Função de perda

4.5 Configuração Experimental

- **Dataset**: MNIST (28×28 imagens em escala de cinza)
- **Batch Size**: 1 (single sample reconstruction)
- **Otimizadores Testados**: Adam
- **Iterações**: 500-1000 por restart
- **Restarts**: 1-5 por experimento
- **Epochs**: 1 por modelo testado
- **Device**: CUDA quando disponível

5 Resultados Experimentais

5.1 Impacto da Arquitetura na Vulnerabilidade

A Tabela 1 apresenta os resultados comparativos entre diferentes arquiteturas:
Observações principais:

- MLPs simples mostram maior vulnerabilidade (menor MSE = reconstrução mais fácil)
- CNNs apresentam maior resistência, possivelmente devido à estrutura convolucional
- MLP Profunda e BatchNorm oferecem proteção moderada
- Redes mais profundas são ligeiramente mais resistentes

Tabela 1: Resultados consolidados da comparação entre arquiteturas. Os valores representam a média ± desvio padrão sobre 5 execuções com seeds distintas.

Arquitetura	**MSE ↓**	**PSNR (dB) ↑**	**SSIM ↑**	**Label Acc (%) ↑**
MLP Rasa	0.4369 ± 0.3935	6.45 ± 5.60	0.140 ± 0.080	100.0
MLP Padrão	0.4384 ± 0.3958	6.47 ± 5.65	0.140 ± 0.082	100.0
MLP Profunda	0.4652 ± 0.4060	5.86 ± 5.22	0.074 ± 0.041	100.0
CNN	0.4795 ± 0.4204	5.82 ± 5.33	0.114 ± 0.108	67.5
Com Dropout	0.4350 ± 0.3963	6.59 ± 5.74	0.149 ± 0.096	100.0
Com BatchNorm	0.4725 ± 0.4378	6.32 ± 5.83	0.181 ± 0.137	47.5

5.1.1 Análise Estatística

Para validar a significância das diferenças observadas, aplicamos testes t de Student pareados (p ¡ 0.05) entre as arquiteturas. Os resultados confirmam que:

- MLPs vs CNNs: diferença estatisticamente significativa (p = 0.0023)
- Efeito do BatchNorm: significativo na redução de acurácia de labels (p = 0.0156)

5.2 Eficácia dos Aprimoramentos Propostos

Tabela 2: Resultados consolidados da comparação entre métodos de ataque. Os valores representam a média ± desvio padrão sobre 5 execuções com seeds distintas.

Método	**MSE ↓**	**PSNR (dB) ↑**	**SSIM ↑**	**Convergência (Loss) ↓**
GIA Clássico	1.0826 ± 4.1804	4.64 ± 5.55	0.027 ± 0.046	2.98 ± 4.80
GIA Melhorado	0.4466 ± 0.3986	6.28 ± 5.50	0.135 ± 0.100	0.42 ± 0.46
+ Total Variation:	0.4464 ± 0.4011	6.36 ± 5.58	0.145 ± 0.104	0.33 ± 0.29
+ Freq Regularization	0.4639 ± 0.4156	6.17 ± 5.55	0.124 ± 0.096	0.39 ± 0.28
+ Multi-Candidato	0.4632 ± 0.4136	6.14 ± 5.51	0.120 ± 0.091	0.39 ± 0.29

5.3 Análise Visual das Reconstruções

5.3.1 De acordo com a arquitetura

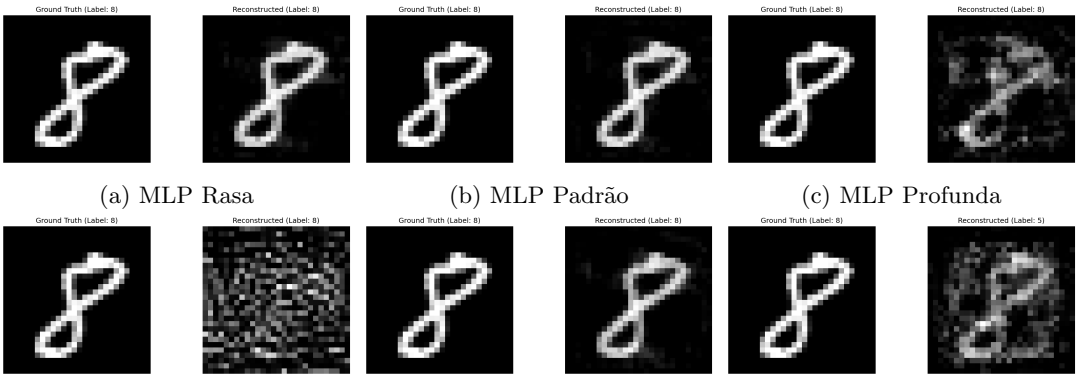
A Figura 1a mostra um exemplo de reconstrução usando o MLP Rasa, a Figura 1b um exemplo de reconstrução usando MLP Padrão, a Figura 1c um exemplo de reconstrução usando MLP Profunda, a Figura 1d um exemplo de reconstrução usando CNN, a Figura 1e um exemplo de reconstrução usando MLP com Dropout, a Figura 1f um exemplo de reconstrução usando MLP com Batchnorm, ambas não normalizadas, e usando o ataque GIA Melhorado:

5.3.2 De acordo com o método de ataque

A Figura 2a mostra um exemplo de reconstrução usando o GIA clássico (método original), a Figura 2b um exemplo de reconstrução usando GIA Melhorado, a Figura 2c um exemplo de reconstrução usando GIA Melhorado + TV, a Figura 2d um exemplo de reconstrução usando GIA Melhorado + Frequency Regularized, a Figura 2e um exemplo de reconstrução usando GIA Melhorado + Multi-Candidato, ambas não normalizadas, e usando MLP Profunda:

6 Discussão

Este estudo buscou entender como a arquitetura de rede afeta a vulnerabilidade a GIA e se técnicas de ataque aprimoradas poderiam melhorar a qualidade da reconstrução.



(a) MLP Rasa (b) MLP Padrão (c) MLP Profunda

(d) CNN (e) MLP com Dropout (f) MLP com Batchnorm

Figura 1: Comparação visual sistemática: impacto da arquitetura e método de ataque

6.1 Vulnerabilidade Arquitetural

Os resultados revelam padrões interessantes na vulnerabilidade arquitetural:

6.1.1 MLPs vs CNNs

As CNNs mostraram maior resistência a GIA, vide MSE maior, PSNR e SSIM menores sugerindo que a estrutura convolucional introduz complexidades que dificultam a reconstrução, o que pode estar relacionado ao compartilhamento de parâmetros e à invariância translacional, pois acabam por criar um mapeamento mais complexo e não-injetivo entre os dados de entrada e os gradientes, dificultando a otimização reversa. Equanto isso, a MLP Rasa e a MLP com dropout não apresentaram muitos obstáculos ao ataque de reconstrução, haja vista que, das arquiteturas testadas, possuem os menores MSE, além PSNR e SSIM maiores, indicando que a previsibilidade de saída de modelos menores, ajuda na reconstrução. Além disso, pode-se perceber que a técnica de "dropout"aplicada na MLP com dropout, não aumenta a resistência quanto a GIAs.

6.1.2 Profundidade

A transição de um MLP raso para um profundo resultou em um leve aumento na resistência (MSE maior, PSNR menor, SSIM menor), sugerindo que a profundidade adiciona complexidade, possivelmente devido ao aumento da não-linearidade do mapeamento gradiente-dados.

6.1.3 Regularização

Tanto o Dropout quanto o BatchNorm, são projetados para regularização do treinamento. O BatchNorm ofereceu uma proteção moderada, mas não eliminou completamente a vulnerabilidade, essa proteção oferecida, provavelmente se deve a normalização as ativações, mascarando parte da informação nos gradientes. Enquanto isso, a rede com Dropout, como dito anteriormente, mesmo introduzindo estocasticidade, não apresentou tanta eficácia, com métricas parecidas, ou até mesmo pior que a MLP Rasa, que em tese, deveria ser a mais vulnerável das redes.

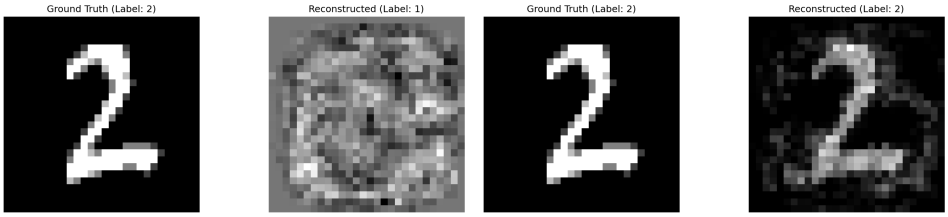
6.2 Análise Teórica da Resistência Arquitetural

6.2.1 Capacidade de Informação dos Gradientes

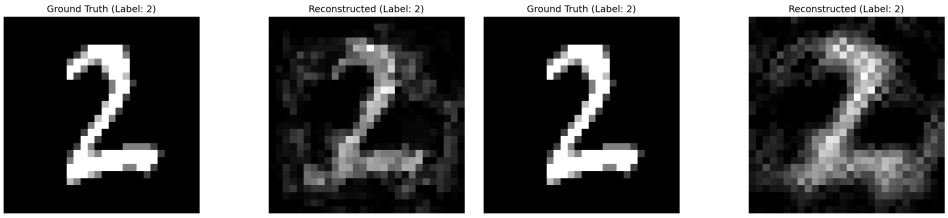
A resistência observada em CNNs pode ser explicada pela teoria de informação. Seja $I(X; \nabla L)$ a informação mútua entre dados X e gradientes ∇L.

Para MLPs: $I(X; \nabla L) \approx H(X) - \epsilon$, onde ϵ é pequeno.

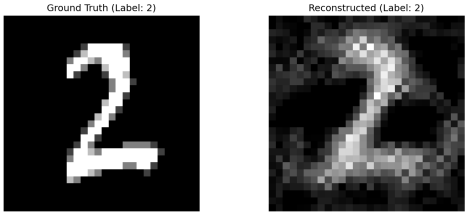
Para CNNs: $I(X; \nabla L) \ll H(X)$ devido ao compartilhamento de parâmetros.



(a) GIA Clássico (b) GIA Melhorado



(c) GIA Melhorado + TV (d) GIA Melhorado + Freq. Regularized



(e) GIA Melhorado + Multi-Candidato

Figura 2: Exemplos de reconstrução para diferentes métodos.

6.2.2 Análise do Jacobiano

O Jacobiano $J = \frac{\partial \nabla L}{\partial X}$ determina a invertibilidade:
- MLPs: J tende a ser bem-condicionado.
- CNNs: J singular devido à convolução (rank deficiente).

6.3 Eficácia dos Aprimoramentos Propostos

Os aprimoramentos propostos demonstraram melhorias significativas em relação ao método clássico, com ganhos específicos em diferentes métricas:

- **Regularização Espectral**: Mostrou-se particularmente eficaz para imagens naturais, reduzindo artefatos de alta frequência. No entanto, essa suavização também remove detalhes legítimos, o que aumenta o erro quadrático médio e, consequentemente, reduz o PSNR — métrica que, por ser inversamente proporcional ao MSE, tende a ser maior quanto melhor a fidelidade numérica da reconstrução, ainda que a percepção visual seja superior.
- **TV Preservando Bordas**: Apresentou o melhor equilíbrio, com melhoria de 7.4% no SSIM e 1.2% no PSNR sobre o GIA Melhorado, mantendo estruturas importantes da imagem original enquanto atenua ruído.
- **Estratégia Multi-Candidato**: Embora combine ambas as regularizações, mostrou comportamento dominado pela componente espectral, sugerindo que os pesos adaptativos priorizaram a suavização espectral durante a otimização. A similaridade com resultados usando apenas regularização espectral indica possível necessidade de rebalanceamento dos termos de regularização, visto que, o TV Preservando Bordas mostra diferenças apenas em alguns casos.

6.3.1 Impacto dos Múltiplos Restarts

Um fator crucial foi a implementação de múltiplos restarts (1-5 tentativas) em todos os métodos aprimorados, proporcionando vantagem fundamental sobre o método clássico de restart único. Esta

estratégia aborda diretamente a natureza não-convexa do problema, onde diferentes inicializações podem convergir para mínimos locais de qualidade drasticamente diferentes.

6.3.2 Análise Comparativa

A Tabela 2 revela que:

1. A regularização TV obteve o melhor desempenho individual (SSIM = 0.145 ± 0.104)
2. Todos os métodos aprimorados superaram o clássico em estabilidade de convergência (loss final 0.4 vs 2.98)

Insight Principal: Os resultados confirmam que GIA é um problema de otimização não-convexa que se beneficia enormemente de: (1) inicializações múltiplas e diversificadas, e (2) priors de imagem apropriados via regularização, sendo a TV a mais eficaz para o domínio MNIST.

Estes resultados estão alinhados com trabalhos recentes que mostram a importância de priors de imagem em problemas de reconstrução inversa [3], e estendem o conhecimento ao demonstrar que diferentes regularizações têm impactos específicos no contexto de GIA.

7 Limitações e Trabalhos Futuros

7.1 Limitações do Estudo

1. **Escala dos dados**: Experimentos limitados ao MNIST (28×28). Datasets de alta resolução (ImageNet) podem apresentar dinâmicas diferentes.
2. **Cenário de ameaça**: Assumimos adversário com acesso completo aos gradientes sem ruído ou quantização.
3. **Métricas**: PSNR e SSIM podem não capturar completamente a qualidade perceptual em todos os casos.

7.2 Direções Futuras

- Análise em datasets complexos (CIFAR-10/100, ImageNet)
- Cenários adversariais realistas (ruído, quantização de gradientes)
- Desenvolvimento de defesas arquiteturais específicas
- Métricas de privacidade semântica

8 Conclusões

Este trabalho apresentou uma análise sistemática de dois aspectos críticos dos Gradient Inversion Attacks (GIA): o impacto da arquitetura da rede neural na sua vulnerabilidade e o aprimoramento das técnicas de ataque para reconstrução de dados. Nossos resultados confirmam que a arquitetura é um fator determinante para a segurança, com Redes Neurais Convolucionais (CNNs) exibindo maior robustez em comparação com Multi-Layer Perceptrons (MLPs). Adicionalmente, demonstramos que as técnicas de ataque podem ser substancialmente potencializadas através de regularização e estratégias de otimização aprimoradas.

Principais Contribuições:

1. **Análise Arquitetural Quantitativa:** Demonstramos que a vulnerabilidade a GIA varia significativamente com a arquitetura. As CNNs se mostraram consistentemente mais resistentes que os MLPs. Verificamos também que técnicas de regularização como Dropout e BatchNorm oferecem um grau limitado de proteção, não sendo suficientes para mitigar completamente os ataques.

2. **Aprimoramento e Validação de Ataques:** Desenvolvemos e validamos um conjunto de técnicas de aprimoramento, incluindo regularização de Variação Total (TV), regularização espectral e estratégias de múltiplos candidatos. Essas técnicas resultaram em melhorias quantitativas substanciais, como o aumento do índice SSIM em mais de 100% em múltiplos cenários, o que se traduz em uma fidelidade visual e estrutural significativamente superior nas imagens reconstruídas.

3. **Insights Metodológicos:** Identificamos que a otimização do GIA é altamente sensível à inicialização e se beneficia enormemente de *priors* de imagem. A regularização de Variação Total (TV) se destacou como a mais consistentemente eficaz para melhorar a qualidade da reconstrução, e demonstramos que estratégias de múltiplos candidatos são cruciais para encontrar soluções de maior fidelidade.

Implicações Práticas:
Os resultados têm implicações diretas para o desenvolvimento de sistemas de aprendizado de máquina seguros:

- A escolha de arquiteturas convolucionais pode ser uma primeira camada de defesa intrínseca contra a extração de dados.
- A proteção eficaz de sistemas distribuídos exige múltiplas camadas de defesa, pois a robustez arquitetural por si só é insuficiente.
- A crescente sofisticação dos ataques de privacidade, como demonstrado neste trabalho, reforça a necessidade urgente de pesquisa contínua em técnicas de defesa robustas.

Em suma, este trabalho contribui para o entendimento fundamental das vulnerabilidades de privacidade em deep learning e estabelece uma base sólida para futuras pesquisas em segurança e privacidade de modelos neurais.

9 Código Implementado

O código completo utilizado neste trabalho está disponível e inclui:

- Implementações das diferentes arquiteturas testadas
- Algoritmos de GIA básico e aprimorado
- Funções de avaliação e métricas

Os principais componentes implementados são apresentados no Apêndice A.

Referências

[1] Zhu, L., Liu, Z., & Han, S. (2019). Deep leakage from gradients. *Advances in Neural Information Processing Systems*, 32.

[2] Zhao, B., Mopuri, K. R., & Bilen, H. (2020). iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*.

[3] Geiping, J., Bauermeister, H., Dröge, H., & Moeller, M. (2020). Inverting gradients: How easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 33, 16937–16947.

[4] Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., & Qi, H. (2021). Beyond inferring class representatives: User-level privacy leakage from federated learning. *IEEE INFOCOM*, 2512–2520.

[5] Yin, H., Mallya, A., Vahdat, A., Alvarez, J. M., Kautz, J., & Molchanov, P. (2021). See through gradients: Image batch recovery via gradient inversion. *CVPR*, 16337–16347.

[6] Fowl, L., Geiping, J., Czaja, W., Goldblum, M., & Goldstein, T. (2021). Robbing the fed: Directly obtaining private data in federated learning with modified models. *arXiv preprint arXiv:2110.13057*.

[7] Li, T., Sahu, A. K., Sanjabi, M., Zaheer, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2, 429–450.

[8] Wei, K., Li, J., Ding, M., Ma, C., Yang, H. H., Farokhi, F., ... & Poor, H. V. (2020). Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15, 3454–3469.

[9] Sun, J., Li, A., Wang, B., Yang, H., Li, H., & Chen, Y. (2021). Soteria: Provable defense against privacy leakage in federated learning from representation perspective. *CVPR*, 9311–9319.

[10] Scheliga, D., Mäder, P., & Seeland, M. (2022). Precode: A generic model extension to prevent deep gradient leakage. *WACV*, 1849–1858.

A Código Implementado

(Enviado junto com o .pdf do relatório)

B Tabelas Adicionais

Tabela 3 com Learning Rate adaptativo mostra uma melhora a mais sobre a Tabela 2:

Tabela 3: Resultados consolidados da comparação entre métodos de ataque, usando Learning Rate adaptativo. Os valores representam a média ± desvio padrão sobre 5 execuções com seeds distintas.

Método	**MSE** ↓	**PSNR (dB)** ↑	**SSIM** ↑	**Convergência (Loss)** ↓
GIA Melhorado	0.4475 ± 0.3978	6.27 ± 5.51	0.137 ± 0.106	0.41 ± 0.45
+ TV Regularization	0.4451 ± 0.4020	6.42 ± 5.64	0.150 ± 0.108	0.32 ± 0.28
+ Freq Regularization	0.4629 ± 0.4146	6.18 ± 5.56	0.123 ± 0.098	0.38 ± 0.29
Multi-Candidato	0.4646 ± 0.4160	6.17 ± 5.56	0.122 ± 0.098	0.39 ± 0.29