Visão Computacional

Estratégias para Reduzir Overfitting

Prof. Dr. Denis Mayr Lima Martins

Pontifícia Universidade Católica de Campinas

Objetivos de Aprendizagem

- Compreender como modificação da função objetivo pode prevenir o overfitting.
- Analisar Estratégias de Normalização (Input Norm e BatchNorm) e Inicialização de Peso (Xavier) para garantir um fluxo de gradientes saudável em redes profundas.
- Utilizar Data Augmentation (aumento de dados) através da expansão artificial do conjunto de treinamento
- Gerenciar o Ciclo de Treinamento com Critérios de Validação (Early Stopping), determinando o ponto ótimo de parada com base no monitoramento do erro no conjunto de validação.

Regularização

- Regularização modifica a função objetivo para **adicionar uma penalidade contra a complexidade**, visando reduzir a variância e a capacidade do modelo.
- Forma geral: adicionar penalidades à função de loss para desencorajar que os pesos atinjam valores excessivamente grandes.

$$\text{Loss}_{\text{w,b}} = \mathcal{L}(\text{y}, \hat{\text{y}}) + \lambda ||\text{w}||_2^2$$

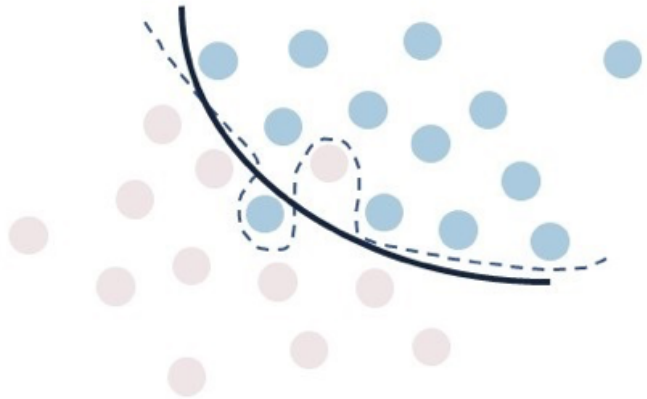
- **L1 (LASSO)**: Usa a norma $||\text{w}||$ (soma dos valores absolutos dos pesos).
- **L2 (Ridge)**: Usa a norma $||\text{w}||_2^2$ (soma dos quadrados dos pesos).

Regularização L1 (exemplo)

```
for data, target in data_loader:
    optimizer.zero_grad()
    output = model(data)
    loss = loss_function(output, target)

    # L1 regularization
    l1_lambda = 0.001
    l1_norm = sum(torch.abs(param).sum() for param in
model.parameters())

    # Combina o termo da L1 reg com a loss original
    total_loss = loss + l1_lambda * l1_norm

    total_loss.backward()
    optimizer.step()
```

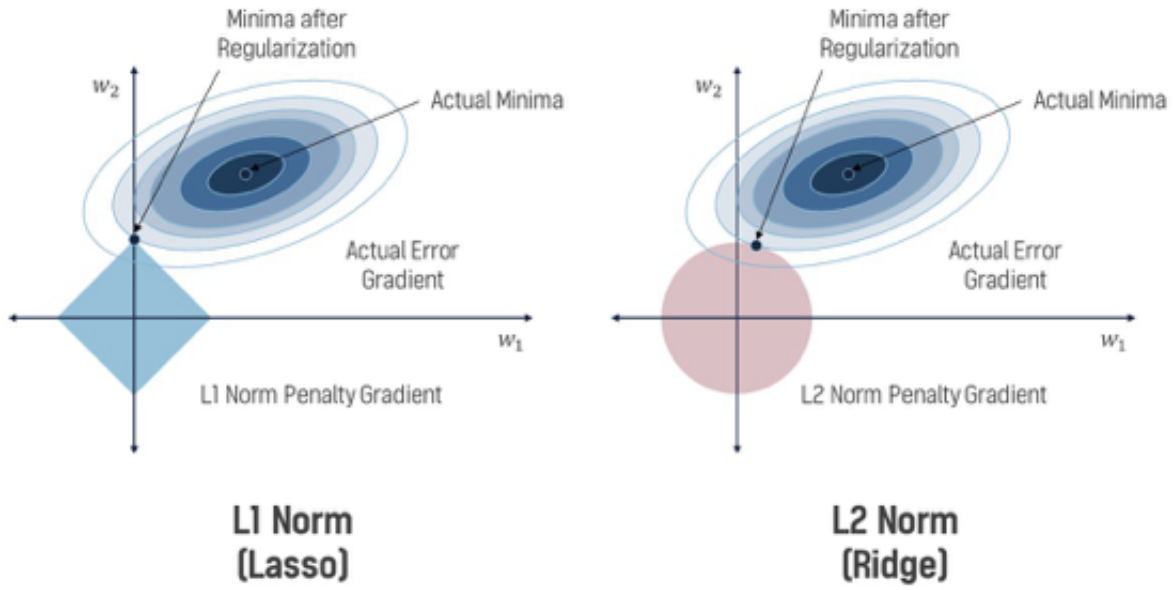
Regularização L2 (exemplo)

```
LAMBDA = 0.01
optimizer =
torch.optim.SGD(
    model.parameters(),
    lr=0.1,
    weight_decay=LAMBDA
)
```



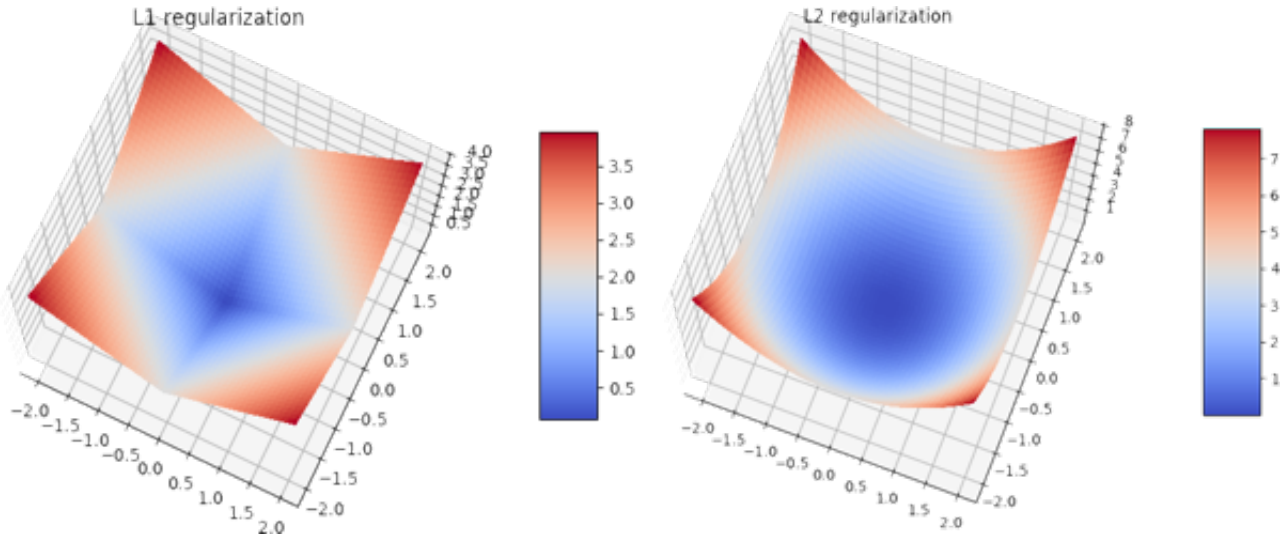
Efeito da Regularização L2. Fonte: Ha-Yoon Song.

Regularização L1 e L2



Efeito da Regularização na otimização por Gradiente Descendente. Fonte: Ha-Yoon Song.

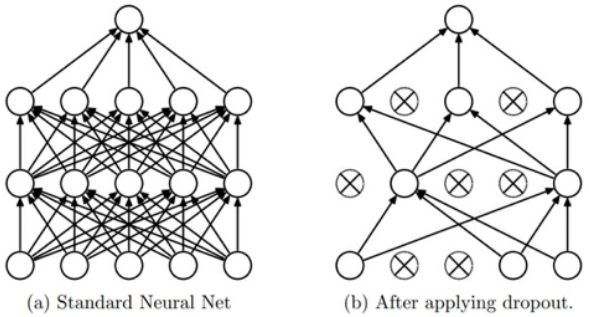
Regularização L1 e L2



Efeito da Regularização na otimização por Gradiente Descendente. Fonte: Juan Miguel Valverde.

Dropout

- O Dropout é uma técnica de regularização que envolve o **descarte aleatório de unidades (neurônios)** (e suas conexões) durante o treinamento.
- Durante o treinamento, cada unidade oculta é mantida com probabilidade $1 - p$ (onde p é a probabilidade de descarte, tipicamente um valor baixo ≈ 0.2). Isso cria um "modelo diferente" para cada minibatch.
- Geralmente aplicadas depois de camadas FC.



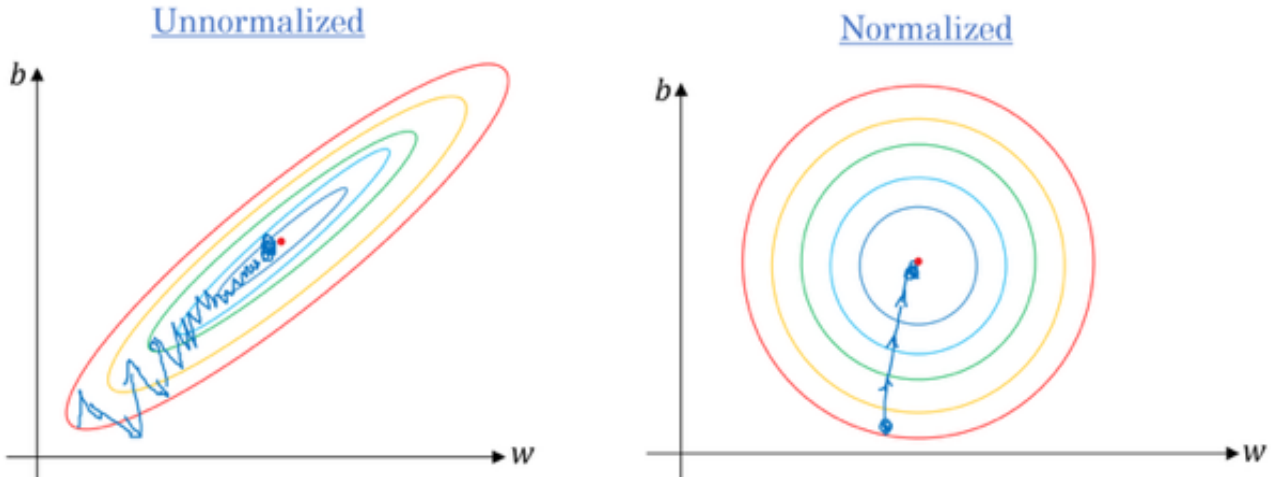
Efeito de Dropout. Fonte: Research Gate.

Dropout

```
class CNNWithDropout(nn.Module):
   def __init__(self):
       super(CNNWithDropout, self).__init__()
       self.my_network = torch.nn.Sequential(
            # ... Camada Linear ...
            torch.nn.ReLU(),
            torch.nn.Dropout(drop_proba),
            # ...
        )
```

Normalização para Estabilização do Treinamento

Input Normalization: Normalizar as features de entrada é um passo inicial crucial para o Gradiente Descendente, garantindo que todas as características tenham zero média e unidade de variância (standardization).



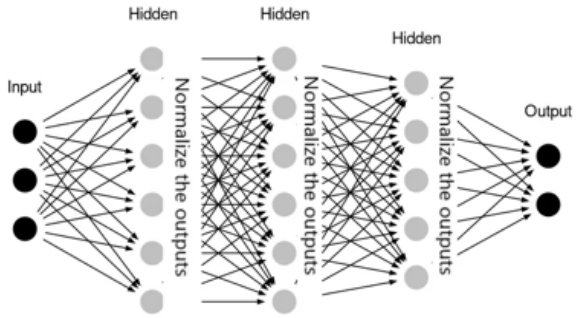
Efeito de Feature Scaling no Gradiente Descendente. Fonte: Quy's blog.

Normalização para Estabilização do Treinamento

Batch Normalization (BatchNorm): Propõe estender a normalização para as entradas das camadas ocultas. A BatchNorm ajuda a lidar com os problemas de **gradientes explosivos/desvanecentes** e melhora a estabilidade e a taxa de convergência do treinamento.

Exemplo:

```
self.conv1 = nn.Conv2d(3, 16,
kernel_size=3, padding=1)
self.bn1 = nn.BatchNorm2d(16)
self.relu = nn.ReLU()
```

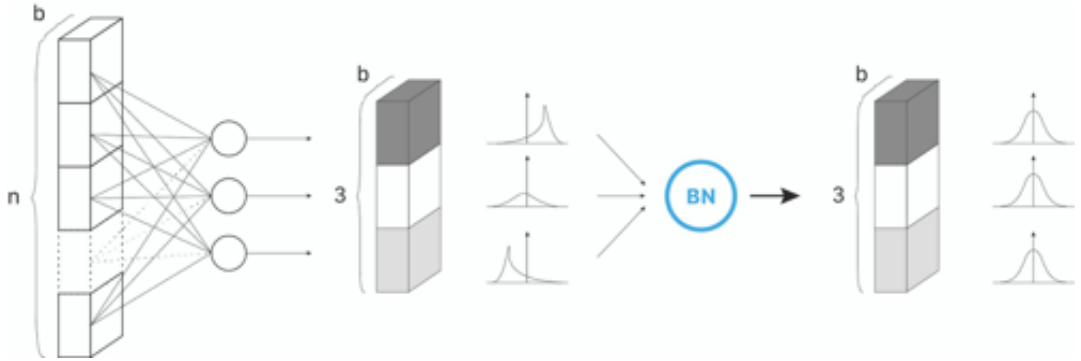


BatchNorm em MLP. Fonte: Ilango Rajagopal.

BatchNorm

Seja uma *net input* $z_j^{[i]}$ do exemplo i para o neurônio j no minibatch de tamanho n:

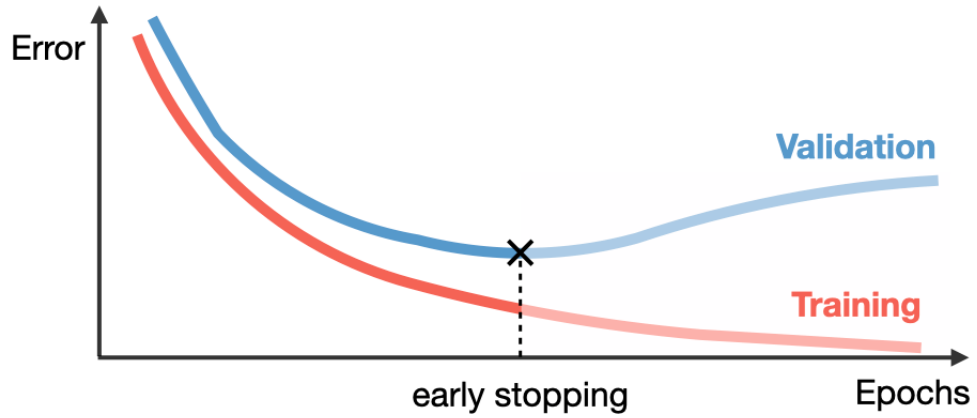
- **Passo 1 (Cálculo de Média e Variância)**:
  $\mu_j = \frac{1}{n}\sum_i z_j^{[i]} \quad \sigma_j^2 = \frac{1}{n}\sum_i (z_j^{[i]} - \mu_j)^2$
- **Passo 2 (Normalização)**: $z_j'^{[i]} = \frac{z_j^{[i]} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$
- **Passo 3 (*Pre-Activation Scaling*)**: $a_j'^{[i]} = \gamma_j \cdot z_j'^{[i]} + \beta_j$
  - γ_j (controla a escala/dispersão)
  - β_j (controla a média) são parâmetros aprendíveis.



Efeito da BatchNorm. Fonte: Towards Data Science.

Early Stopping

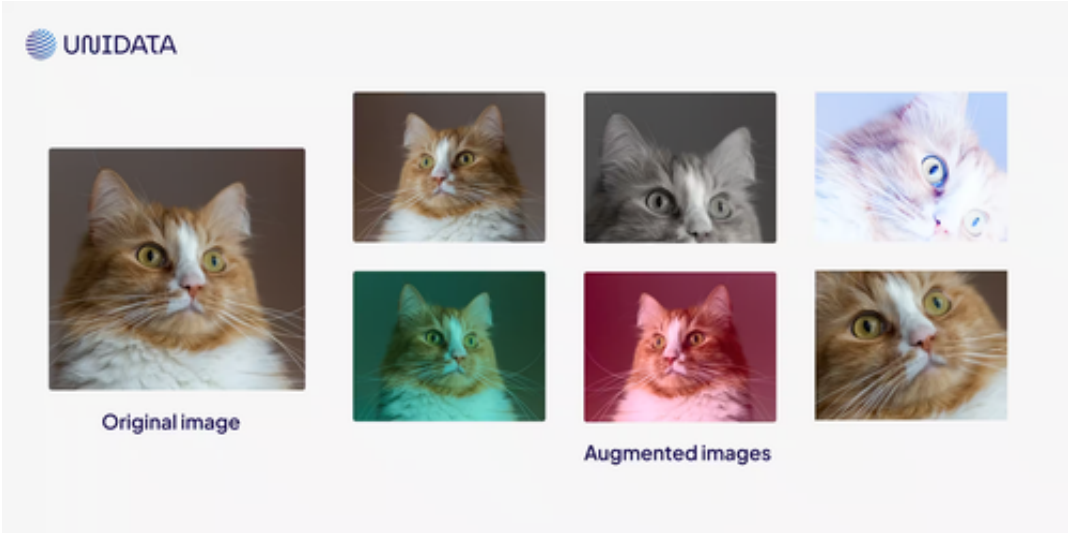
- Modelos complexos tendem a ter um desempenho de generalização (validação) que melhora inicialmente e depois se degrada à medida que o *overfitting* começa.
- O *Early Stopping* é uma estratégia de regularização que envolve **parar o treinamento** antes da convergência total no conjunto de treinamento.
- **Critério:** O treinamento é interrompido quando a métrica de desempenho no conjunto de validação (geralmente a perda de validação) para de melhorar ou começa a piorar significativamente por um número de épocas pré-definido (*patience*).



Early Stopping. Fonte: wandb.ai.

Data Augmentation

- Gerar artificialmente mais dados de treinamento a partir do conjunto existente. Isso é frequentemente a melhor maneira de reduzir o *overfitting* quando não é viável coletar mais dados reais.
- Aumento de dados injeta invariância e robustez ao modelo, expondo-o a variações que preservam a classe, como: **Flip** (Inversão Horizontal), **Crop** (Corte Aleatório), **Color Jitter** (Aumento de cor/brilho), **Rotation** (Rotação Aleatória), etc.



Exemplos de Data Augmentation. Fonte: UniData.

Data Augmentation (exemplo)

```
training_transforms = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Resize(32, 32),
    torchvision.transforms.RandomCrop(28, 28),
    torchvision.transforms.RandomRotation(degrees=30),
    torchvision.transforms.Normalize(mean=(0.5), std=(0.5))
])
```

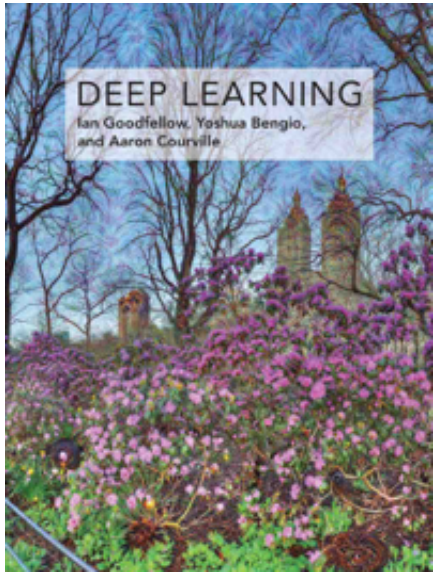
Inicialização de Pesos

- Uma inicialização inadequada dos pesos pode levar a problemas de **gradientes desvanecentes/explosivos** (Vanishing/Exploding Gradient Problems), especialmente em redes profundas. Inicialmente, buscava-se usar pequenos números aleatórios para quebrar a simetria.
- **Inicialização Xavier/Glorot**: Proposta para funções de ativação como TanH. O método escala os pesos em proporção ao número de entradas na camada (*fan in*).
- **Inicialização He/Kaiming**: Desenvolvida para ativações ReLU. PyTorch utiliza o esquema Kaiming He por padrão.
- **Leitura Recomendada**: A Gentle Introduction To Weight Initialization for Neural Networks

> *Para uma intuição visual sobre o efeito da inicialização dos parâmetros, veja: https://www.deeplearning.ai/ai-notes/initialization/index.html*

Resumo

- **Regularização:** O objetivo é penalizar a complexidade do modelo. Isso inclui a **Regularização L1/L2**, que impõe a contração dos pesos, e o **Dropout**, que previne a co-adaptação de características ao descartar unidades aleatoriamente, simulando um método de *ensemble*.
- **Inicialização de Pesos** é crucial para garantir que a variância das ativações e dos gradientes se mantenha estável em profundidade, prevenindo o problema de gradiente desvanecente. **BatchNorm** complementa isso, estabilizando as entradas das camadas ocultas e aumentando a taxa de convergência.
- **Data Augmentation** expande artificialmente o conjunto de treinamento, aumentando a robustez do modelo a variações de entrada.
- **Early Stopping** atua no domínio temporal, usando o dataset de validação para interromper o treinamento no ponto de máxima generalização, limitando a capacidade efetiva da rede.



Leitura Recomendada:
Capítulo 7.

Perguntas e Discussão

- Qual é a diferença fundamental no mecanismo de combate ao *overfitting* entre a **Regularização L1/L2** (penalidade de norma, forçando pesos menores) e o **Dropout** (desativação estocástica de neurônios, prevenindo a co-adaptação)? Em que cenários cada abordagem tende a ser mais eficaz?
- A **Normalização de Batch (BatchNorm)** provou ser um método robusto para estabilizar o treinamento e acelerar a convergência. Discuta como a BatchNorm reduz a dependência de esquemas precisos de **inicialização de pesos** (como He ou Xavier) e por que a inicialização ainda permanece relevante, mesmo com a presença de BatchNorm.
- Se as curvas de treinamento e validação de um modelo divergem significativamente (indicando alta variância/overfitting), como um engenheiro de *Deep Learning* decidiria a ordem de prioridade entre: **a) aumentar a Regularização L2**, **b) aumentar a complexidade do pipeline de Data Augmentation**, e **c) reduzir a paciência do Early Stopping**?