A3: Training robust neural networks

Alexandros Kouvatseas | Luka Lafaye de Micheaux | Matthieu Neau

Attack Techniques

- PGD - Projected Gradient Descent
- FGSM - Fast Gradient Sign Method
- MIM - Momentum Iterative Method

PGD – Projected Gradient Descent

Algorithm 1 PGD Attack

Input: Model $model$, images X, labels Y
Parameters: Perturbation ϵ, step size α, iterations N, device $device$
Output: Perturbed images X'

$X' \leftarrow X$
Enable gradient computation for X'
for $i \leftarrow 1$ to N **do**
    $outputs \leftarrow model(X')$
    $loss \leftarrow F.nll\_loss(outputs, Y)$
    Reset gradients: $model.zero\_grad()$
    Compute gradients: $loss.backward()$
    $X' \leftarrow X' + \alpha \cdot \text{sign}(X'.grad)$
    Clip X' within $[X - \epsilon, X + \epsilon]$ and $[0, 1]$
    Detach X' from the current graph
    Re-enable gradient computation for X'
end for
return X'

Strengths

- Theoretically grounded in constrained optimization
- Iterative Refinement

Weaknesses

- Computationally intensive
- Sensitive to hyperparameters
- Overfitting risk

FGSM

Algorithm 2 FGSM attack

Input: Neural network $model$, images X, labels Y
Parameters: Perturbation ϵ, computation device $device$
Output: Perturbed images X'

Enable gradient computation for X
$outputs \leftarrow model(X)$
$loss \leftarrow F.nll\_loss(outputs, Y)$
$model.zero\_grad()$
Compute gradients: $loss.backward()$
$X' \leftarrow X + \epsilon \cdot \text{sign}(X.grad)$
Clip X' to be within valid pixel range [0, 1]
return X'

Strengths

- Fast with a single step
- Simple to implement

Weaknesses

- Sensitive to ε
- Mainly designed for $\ell\infty$ bounded perturbations

MIM

Algorithm 3 MIM Attack

Input: Model $model$, images X, labels Y
Parameters: Max perturbation ϵ, step size α, iterations N, momentum μ, device $device$
Output: Adversarially perturbed images X'

Initialize $g \leftarrow \mathbf{0}$ (same shape as X)
for $i \leftarrow 1$ to N **do**
    Enable gradient computation for X'
    $outputs \leftarrow model(X')$
    $loss \leftarrow F.nll\_loss(outputs, Y)$
    Compute gradients: $grad \leftarrow$ autograd.grad$(loss, X')[0]$
    Normalize gradients: $grad\_norm \leftarrow$ torch.norm$(grad.view(grad.shape[0], -1), p = 1, dim = 1)$
    $grad\_normalized \leftarrow grad/(grad\_norm.view(-1, 1, 1, 1) + 1e - 8)$
    Update momentum: $g \leftarrow \mu \cdot g + grad\_normalized$
    $X' \leftarrow X' + \alpha \cdot \text{sign}(g)$
    Clip change: $delta \leftarrow$ torch.clamp$(X' - X, min = -\epsilon, max = \epsilon)$
    Clip X': $X' \leftarrow$ torch.clamp$(X + delta, min = 0, max = 1)$
    Detach X' from computation graph
end for
return X'

Strengths

- Incorporates momentum to stabilize updates
- Reduces oscillations in gradient-based optimization

Weaknesses

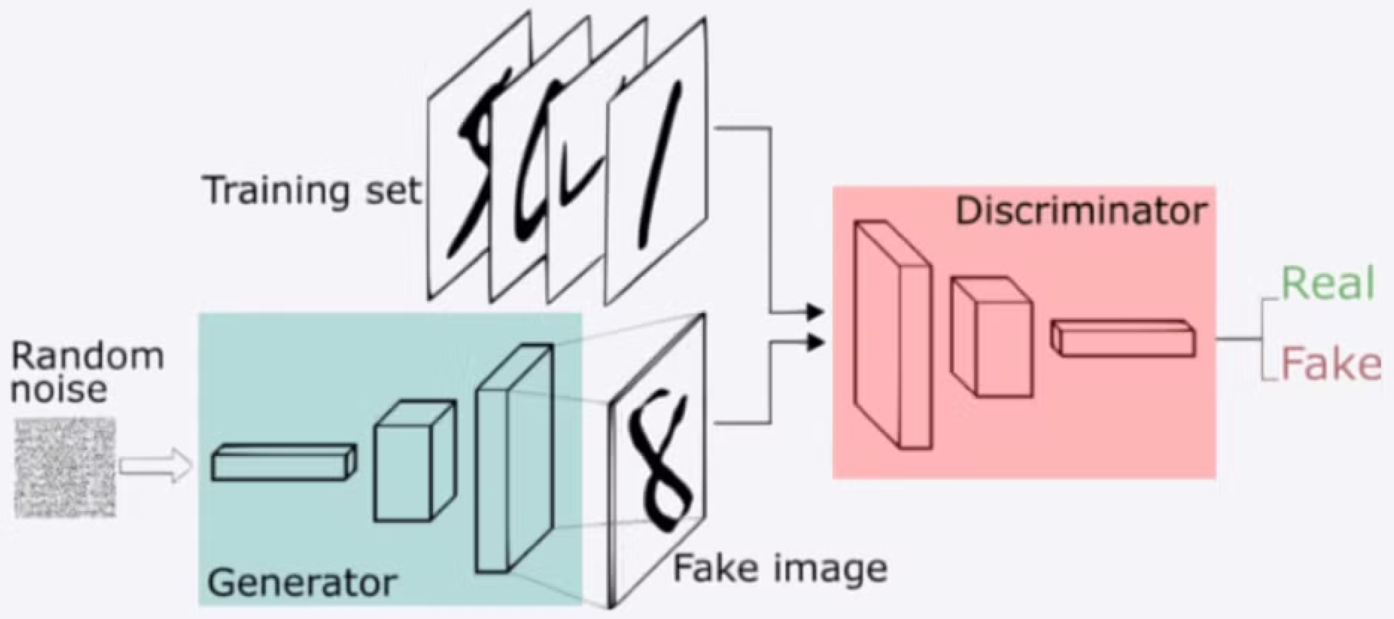
- Computationally intensive
- Sensitive to momentum decay

Defense Techniques

- Adversarial Training - enhances resistance to attacks but reduces clean accuracy

- Regularization Techniques - maintains stable predictions under small perturbations

- Multi-attack technique (adversarial training using all attacks)



$$\mathcal{L}_{\text{total}} = \mathcal{L}(f(x), y) + \lambda \|\nabla_x \mathcal{L}(f(x), y)\|_p^2$$

$\mathcal{L}(f(x), y)$: Original loss function.

$\|\nabla_x \mathcal{L}(f(x), y)\|_p^2$: Regularization term penalizing large gradients

Adversarial Training

- Trains the model on modified data points known as adversarial examples.
- **Objective of Adversarial Examples**: Introduce errors into the model's predictions, aiming to maximize the prediction error during training to make the model more robust.
- **Balanced Training Approach**: Adjusts the training process to include a mix of both natural and adversarially altered inputs, with the aim of minimizing overall prediction errors and enhancing model resilience.

Regularization Techniques (1)

- **Spectral Normalization**: This technique adjusts each layer's weights by dividing them by their largest singular value, effectively moderating the layer's sensitivity to input perturbations.

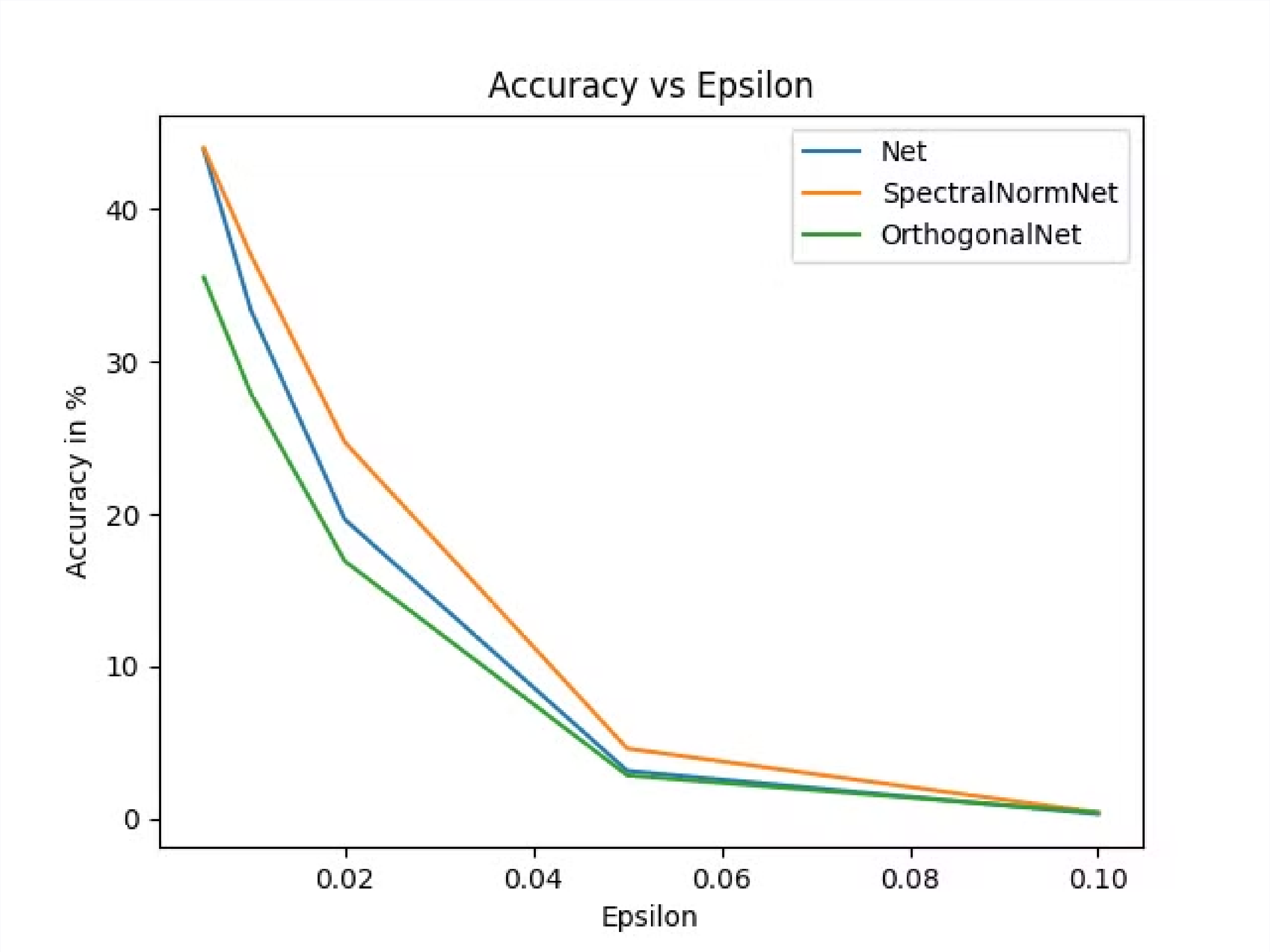
$$\sigma(a) = \max_{\mathbf{h}:\mathbf{h}\neq 0} \frac{\|A\mathbf{h}\|_2}{\|\mathbf{h}\|_2} = \max_{\|\mathbf{h}\|_2\leq 1} \|A\mathbf{h}\|_2$$

$$\bar{W}_{\mathrm{SN}}(W) = W/\sigma(W)$$

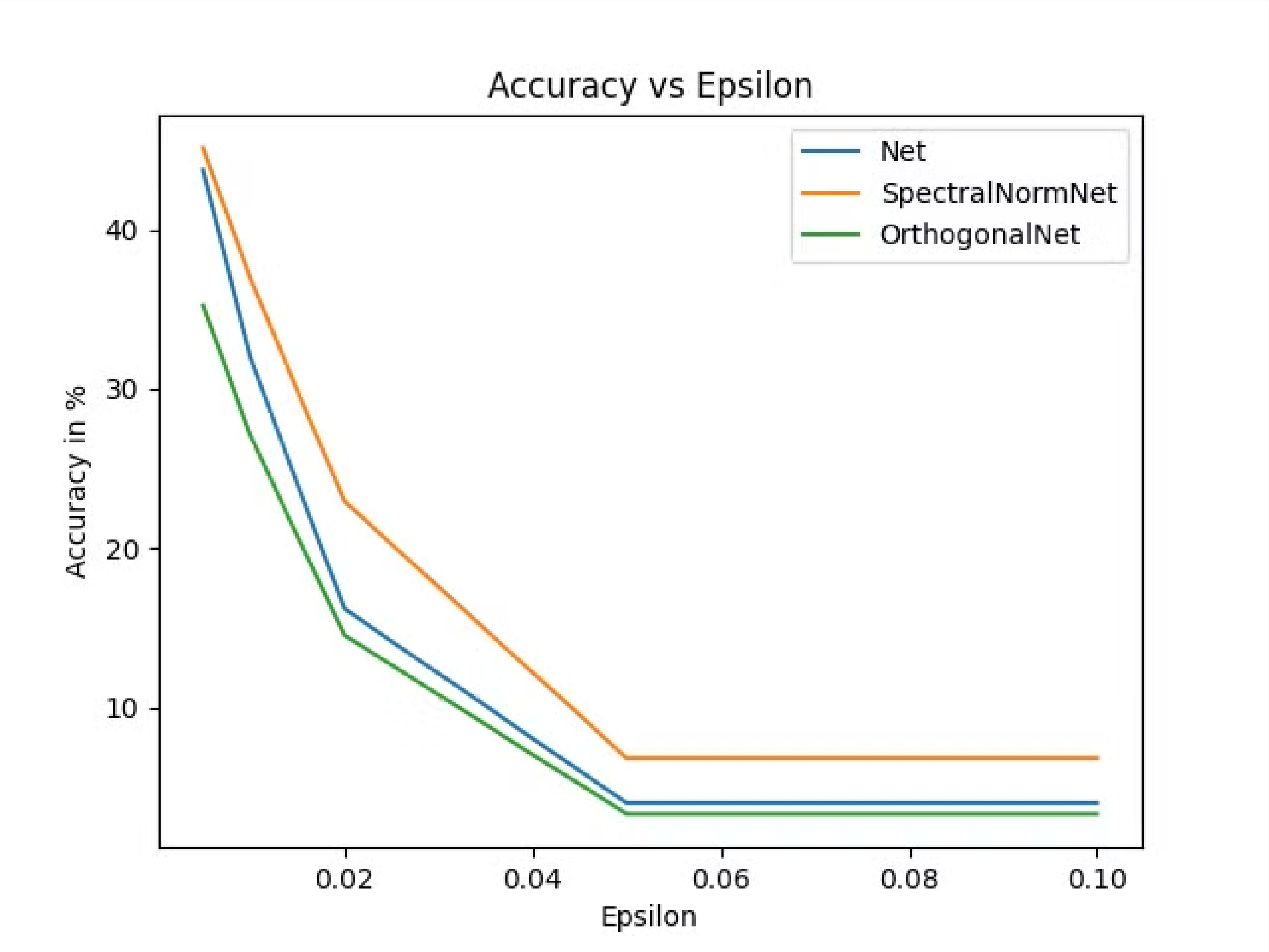
- **Orthogonal Normalization**: Attempts to maintain weight matrices close to orthogonality to stabilize learning, though found less effective for tasks unrelated to disentangling latent spaces.

$$\mathcal{L}_{ortho} = \sum(|WW^T - I|)$$

Regularization Results (1)



FGSM attacks



PGD attacks, alpha = 0.001, n_iter = 40

Regularization Techniques (2)

- **Gradient Regularization:** Combines adversarial training with a gradient penalty to counteract the effects of input perturbations, improving model robustness against adversarial attacks

$$P = \frac{1}{N} \sum_{i=1}^{N} \left(\left\| \nabla_{\mathbf{x}_i'} L \right\|_2 \right)^2$$

Multi-Attack

- **Multi-Attack Technique**: Multiple adversarial attacks (FGSM, MIM, PGD) are applied in a cyclical manner during training, enhancing model robustness by exposing it to varied perturbations.

- **Cyclical Application**: Each training batch applies a different attack based on the batch index, ensuring even exposure to all attack types throughout the training epochs.

- **Diverse Input Utilization**: Utilizing adversarial examples from different attacks as inputs for training prevents model overfitting and contributes to superior performance by increasing input diversity.

Results

Attack	Defense	Nat Accuracy	PGD linf	PGD l2
PGD	Adversarial Training	56.25%	17.26%	26.4%
FGSM	Adversarial Training	50%	0.46%	8.48%
MIM	Adversarial Training	37.5%	17.23%	26.36%
PGD	Gradient Regularization	25%	27.43%	27.01%
PGD/MIM/FGSM	Multi-Attack	43.75%	28.39%	34.87%

Thank you for listening!

Dauphine
UNIVERSITÉ PARIS | PSL