Project Proposal Presentation

Group-6

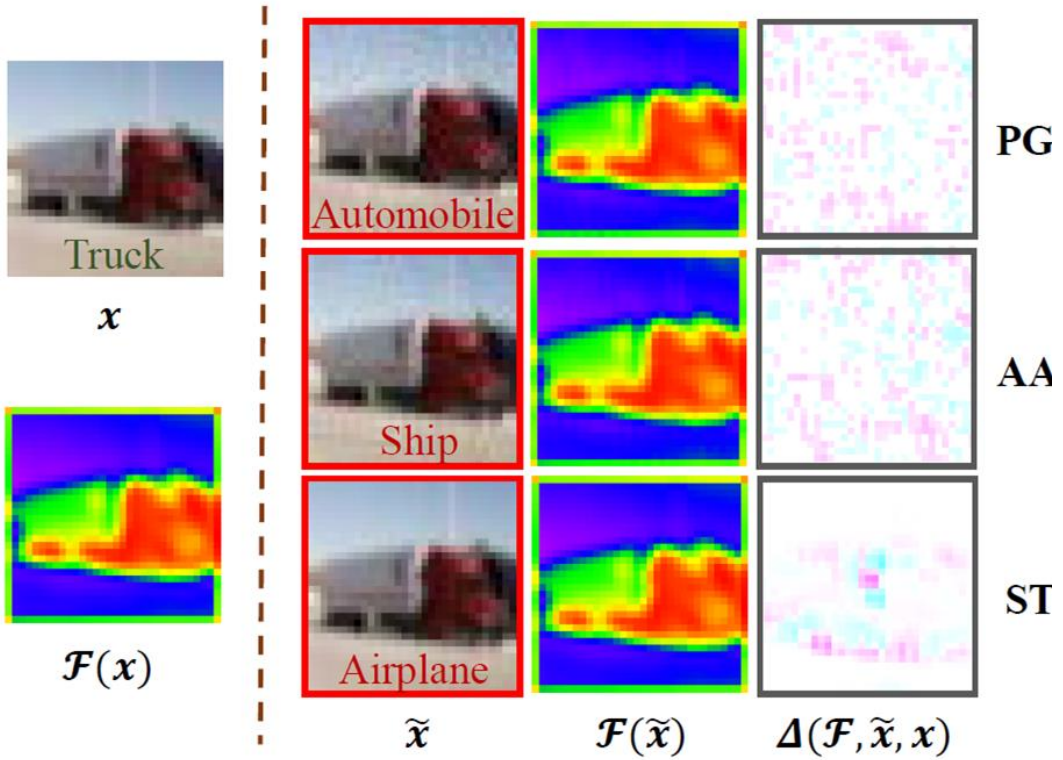
EEL6812 (Spring 2024)

CONTENTS

Adversarial Attacks on Deep Learning

- Is this a serious threat? **Yes**

- Why? Many security-sensitive applications such as face recognition and autonomous driving rely heavily on the accuracy and reliability of machine learning models.

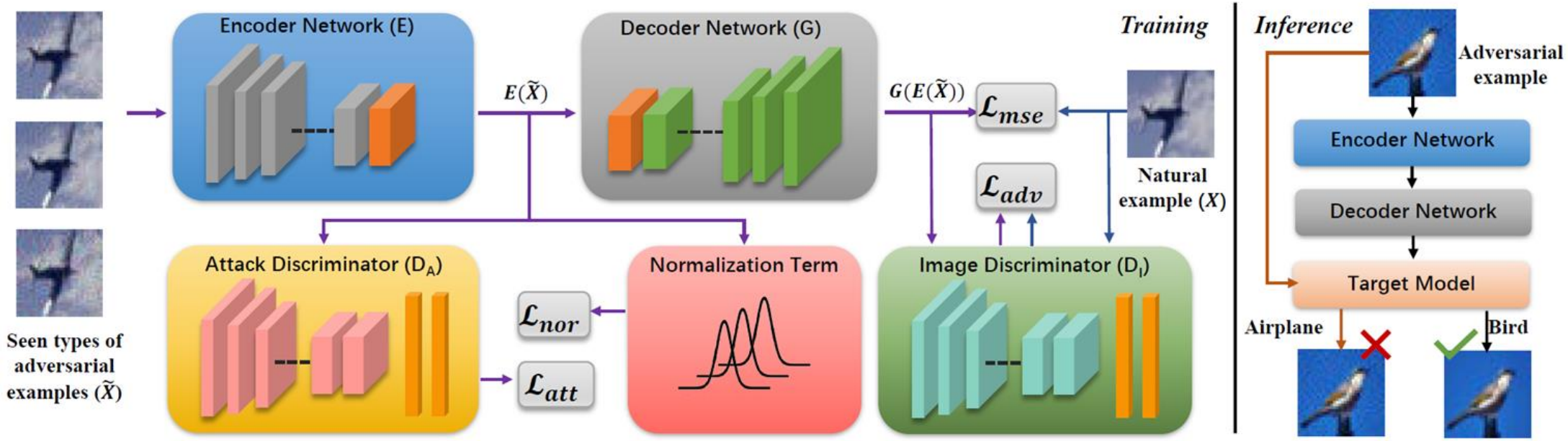


Adversarial Attacks on Deep Learning

Attack techniques	Defense techniques
• AA (auto attack) • PGD (projected gradient descent) • STA (spatial transform attack)	• AT (adversarial training) • ARN (adversarial noise removing network)

Related work/Reference Paper-1

- ARN (adversarial noise removing network)



ARN (adversarial noise removing network)

Encoder

$$\mathcal{L}_{D_A} = -\frac{1}{K}\sum_{k=1}^{K} Y_k^p \cdot \log(\sigma(D_A(E(\tilde{X}_k)))), \quad (1)$$

$$\mathcal{L}_{att} = -\frac{1}{K}\sum_{k=1}^{K} Y_\zeta^p \cdot \log(\sigma(D_A(E(\tilde{X}_k)))). \quad (2)$$

$$\mathcal{L}_{nor} = JSD\left(P_1, \cdots, P_K\right) = \frac{1}{K}\sum_{k=1}^{K} KL\left(P_k \| \mathcal{N}\right). \quad (3)$$

Data is normalized to match multivariate Gaussian prior distribution

Decoder

$$\mathcal{L}_{mse} = \sum_{k=1}^{K} \|G(E(\tilde{X}_k)) - X\|_2^2, \quad (4)$$

where $\|\cdot\|_2$ is the L_2 norm.

$$\mathcal{L}_{D_I} = \sum_{k=1}^{K} [\log(D_I(G(E(\tilde{X}_k)))) + \log(1 - D_I(X))]. \quad (5)$$

$$\mathcal{L}_{adv} = -\sum_{k=1}^{K} \log(D_I(G(E(\tilde{X}_k)))). \quad (6)$$

ARN (adversarial noise removing network)

Loss functions:

$$\mathcal{L}_E = \mathcal{L}_{mse} + \lambda_1 \mathcal{L}_{att} + \lambda_2 \mathcal{L}_{nor}, \tag{7}$$

$$\mathcal{L}_G = \mathcal{L}_{mse} + \theta \mathcal{L}_{adv}, \tag{8}$$

Algorithm 1 ARN: Adversarial Noise Removing Network

Input: Natural examples X and adversarial examples $\tilde{X}$.

repeat

1: Sample a mini-batch X_d and $\tilde{X}_d$ from X and $\tilde{X}$ respectively.
2: Forward-pass $\tilde{X}_d$ through E to obtain encoded features $E(\tilde{X}_d)$ and calculate $\mathcal{L}_{D_A}$ (Eq. 1), $\mathcal{L}_{att}$ (Eq. 2) and $\mathcal{L}_{nor}$ (Eq. 3).
3: Forward-pass $E(\tilde{X}_d)$ through G to restore natural examples and calculate $\mathcal{L}_{mse}$ (Eq. 4), $\mathcal{L}_{adv}$ (Eq. 6) and $\mathcal{L}_{D_I}$ (Eq. 5).
4: Back-pass and update E, G to minimize $\mathcal{L}_E$ (Eq. 7) and $\mathcal{L}_G$ (Eq. 8).
5: Update D_A and D_I to minimize $\mathcal{L}_{D_A}$ (Eq. 1) and $\mathcal{L}_{D_I}$ (Eq. 5).

until E and G converge.

ARN (adversarial noise removing network)

	Defense	Attacks								
		None	PGD_N	PGD_T	CW_N	DDN_N	AA_N	$JSMA_T$	$PGD_{N\epsilon'}$	$AA_{N\epsilon'}$
LeNet	None	**0.64**	100	100	100	100	100	100	100	100
	AT_{PGD}	1.19	9.63	8.38	6.42	5.91	12.60	28.59	54.34	60.06
	$DOA_{7\times7}$	6.27	65.23	38.84	11.48	10.53	68.49	19.81	86.76	92.51
	APE-G_{PP}	1.57	8.76	3.20	2.34	2.15	12.40	36.49	34.86	46.72
	APE-G_{DP}	1.73	10.39	5.81	2.93	1.91	15.26	38.04	37.33	49.38
	HGD_{PP}	1.36	1.89	1.30	1.67	1.54	2.43	50.62	75.79	90.34
	HGD_{DP}	1.18	2.56	1.91	1.79	1.23	3.30	53.73	78.95	93.76
	ARN_{PP}	1.16	**1.85**	**1.29**	**1.45**	1.28	**2.38**	**16.75**	**15.27**	**26.84**
	ARN_{DP}	1.11	1.91	1.80	1.53	**1.22**	2.97	17.81	17.63	29.74
ResNet	None	**7.67**	100	100	100	99.99	100	100	100	100
	AT_{PGD}	12.86	51.02	49.68	50.17	49.19	53.66	44.59	59.09	61.65
	$DOA_{7\times7}$	9.82	89.03	73.96	24.11	49.29	97.52	**23.26**	96.83	97.75
	APE-G_{PP}	23.08	44.38	39.09	23.18	32.39	60.09	39.10	79.34	87.16
	APE-G_{DP}	24.23	45.96	41.50	27.43	24.73	64.82	41.67	83.19	89.92
	HGD_{PP}	10.41	39.44	23.03	13.26	16.02	42.34	38.65	57.97	58.41
	HGD_{DP}	9.42	41.62	25.30	12.46	**10.04**	43.45	43.63	58.63	59.86
	ARN_{PP}	8.21	**38.66**	**20.43**	**11.47**	14.64	**38.94**	35.49	**49.45**	**52.64**
	ARN_{DP}	8.18	40.28	22.87	12.24	10.17	41.27	36.23	52.87	55.91

Limitations of ARN

- Adversarial transferability
- Computational complexity
- Sensitivity to hyperparameters
- Resource requirement for training
- Should handle feature preservation

Our takeaway from this Paper

Can we modify those AIF and create adversarial example??

cation information and ignore the adversarial noise. Note that adversarial examples are designed to retain the invariant features so that we human could not identify the adversarial examples in advance, e.g., by constraining the adversarial noise to be small or non-suspicious (Goodfellow et al., 2015; Gilmer et al., 2018). We name such invariant features as *attack-invariant features* (AIF).

Misclassification Aware Adversarial Training

In traditional adversarial training, methods may not fully account for the contribution of misclassified examples to the model's vulnerability to adversarial attacks.

MART addresses this limitation by introducing a weighted loss function that differentiates between misclassified and correctly classified examples during training.

$$\mathcal{L}^{\text{MART}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{x}_i, y_i, \boldsymbol{\theta}),$$

Loss Function (MART)

$$\ell(\mathbf{x}_i, y_i, \boldsymbol{\theta}) := \mathrm{BCE}\big(\mathbf{p}(\hat{\mathbf{x}}_i', \boldsymbol{\theta}), y_i\big) + \lambda \cdot \mathrm{KL}\big(\mathbf{p}(\mathbf{x}_i, \boldsymbol{\theta}) \| \mathbf{p}(\hat{\mathbf{x}}_i', \boldsymbol{\theta})\big) \cdot \big(1 - \mathbf{p}_{y_i}(\mathbf{x}_i, \boldsymbol{\theta})\big)$$

- Weighted Loss Function
- Misclassification Penalty
- Regularization

Results (MART)

Table 2: White-box robustness (accuracy (%) on white-box test attacks) on MNIST and CIFAR-10.

Defense	MNIST				CIFAR-10			
	Natural	FGSM	PGD^{20}	CW_∞	Natural	FGSM	PGD^{20}	CW_∞
Standard	99.11	97.17	94.62	94.25	84.44	61.89	47.55	45.98
MMA	98.92	97.25	95.25	94.77	**84.76**	62.08	48.33	45.77
Dynamic	98.96	97.34	95.27	94.85	83.33	62.47	49.40	46.94
TRADES	**99.25**	96.67	94.58	94.03	82.90	62.82	50.25	48.29
MART	98.74	**97.87**	**96.48**	**96.10**	83.07	**65.65**	**55.57**	**54.87**

Table 3: Black-box robustness (accuracy (%) on black-box test attacks) on MNIST and CIFAR-10.

Defense	MNIST				CIFAR-10			
	FGSM	PGD^{10}	PGD^{20}	CW_∞	FGSM	PGD^{10}	PGD^{20}	CW_∞
Standard	96.12	95.73	95.47	96.34	79.98	80.27	80.01	80.85
MMA	96.11	95.94	95.81	96.87	80.28	80.52	80.48	81.32
Dynamic	97.60	96.25	95.82	97.03	81.37	81.71	81.38	82.05
TRADES	97.49	96.03	95.73	97.20	81.52	81.73	81.53	82.11
MART	**97.77**	**96.96**	**96.97**	**98.36**	**82.75**	**82.93**	**82.70**	**82.95**

Limitations of MART

- Sensitivity to hyperparameters
- Vulnerable to adaptive attacks
- Scalability
- Computational Overhead

Probabilistic Margins for Instance Reweighting in Adversarial Training

Existing methodology:

Reweighting adversarial data during training has been recently shown to improve adversarial robustness, where data closer to the current decision boundaries are regarded as more critical and given larger weights.

Limitation:

The closer the data is to the current boundary, higher the weightage given to the data. So the object closer to bounding box is taken as critical data point and given higher importance.

Probabilistic Margins for Instance Reweighting in Adversarial Training

solution:

PM is employed for reweighting adversarial data during AT, where we propose the Margin-Aware Instance reweighting Learning (MAIL). MAIL pays much attention to those non-robust data.

The authors propose 3 different Probabilistic margin (PM) which are continuous and path independent.

- PM is a geometric measurement from a data point to the closest decision boundary.

$$PM\_nat_i = p_{y_i}(x_i;\theta) - \max_{j;j\neq y_i} p_j(x_i;\theta)$$

$$PM\_adv_i = p_{y_i}(x_i+\delta_i;\theta) - \max_{j;j\neq y_i} p_j(x_i+\delta_i;\theta)$$

$$PM\_diff_i = PM\_nat_i - PM\_adv_i$$

Probabilistic Margins for Instance Reweighting in Adversarial Training

Algorithm:

Algorithm 1 MAIL: The Overall Algorithm.

Input: a network model with the parameters θ; and a training dataset S of size n.
Output: a robust model with parameters θ^*.

for $e = 1$ **to** `num_epoch` **do**
  for $b = 1$ **to** `num_batch` **do**
    sample a mini-batch $\{(x_i, y_i)\}_{i=1}^{m}$ from S; ▷ mini-batch of size m.
    for $i = 1$ **to** `batch_size` **do**
      $\delta_i^{(0)} = \xi$, with $\xi \sim \mathcal{U}(0, 1)$;
      for $t = 1$ **to** T **do**
        $\delta_i^{(t)} \leftarrow \texttt{Proj}\left[\delta_i^{(t-1)} + \alpha \texttt{sign}\left(\nabla_\theta \ell(x_i + \delta_i^{(t-1)}, y_i; \theta)\right)\right]$;
      end for
      $w_i^{\text{unn}} = \texttt{sigmoid}(-\gamma(\texttt{PM}_i - \beta))$;
    end for
    $\omega_i = M \times w_i^{\text{unn}} / \sum_j w_j^{\text{unn}}, \forall i \in [m]$; ▷ $\omega_i = 1$ during burn-in period.
    $\theta \leftarrow \theta - \eta \nabla_\theta \sum_{i=1}^{m} \omega_i \ell(x_i + \delta_i, y_i; \theta) + \mathcal{R}(x_i, y_i; \theta)$;
  end for
end for

Probabilistic Margins for Instance Reweighting in Adversarial Training

Algorithm:

Algorithm 2 MAIL-AT: The Overall Algorithm.

Input: a network model with the parameters θ; and a training dataset S of size n.
Output: a robust model with parameters θ^*.

for $e = 1$ **to** `num_epoch` **do**
  for $b = 1$ **to** `num_batch` **do**
    sample a mini-batch $\{(x_i, y_i)\}_{i=1}^{m}$ from S; ▷ mini-batch of size m.
    for $i = 1$ **to** `batch_size` **do**
      $\delta_i^{(0)} = \xi$, with $\xi \sim \mathcal{U}(0, 1)$;
      for $t = 1$ **to** T **do**
        $\delta_i^{(t)} \leftarrow \texttt{Proj}\left[\delta_i^{(t-1)} + \alpha \texttt{sign}\left(\nabla_\theta - \log \mathrm{p}_{y_i}(x_i + \delta_i^{(t-1)}; \theta)\right)\right]$;
      end for
      $w_i^{\text{unn}} = \texttt{sigmoid}(-\gamma(\texttt{PM}_i - \beta))$;
    end for
    $\omega_i = M \times w_i^{\text{unn}} / \sum_j w_j^{\text{unn}}, \forall i \in [m]$; ▷ $\omega_i = 1$ during burn-in period.
    $\theta \leftarrow \theta - \eta \nabla_\theta \left(-\sum_{i=1}^{m} \omega_i \log \mathrm{p}_{y_i}(x_i + \delta_i^{(T)}; \theta)\right)$;
  end for
end for

Probabilistic Margins for Instance Reweighting in Adversarial Training

Algorithm:

Algorithm 3 MAIL-TRADES: The Overall Algorithm.

Input: a network model with the parameters θ; and a training dataset S of size n.
Output: a robust model with parameters θ^*.

for $e = 1$ **to** `num_epoch` **do**
  for $b = 1$ **to** `num_batch` **do**
    sample a mini-batch $\{(x_i, y_i)\}_{i=1}^{m}$ from S; ▷ mini-batch of size m.
    for $i = 1$ **to** `batch_size` **do**
      $\delta_i^{(0)} = \xi$, with $\xi \sim \mathcal{U}(0, 1)$;
      for $t = 1$ **to** T **do**
        $\delta_i^{(t)} \leftarrow \texttt{Proj}\left[\delta_i^{(t-1)} + \alpha \texttt{sign}\left(\nabla_\theta \texttt{KL}(\mathbf{p}(x_i + \delta_i^{(t-1)}; \theta) || \mathbf{p}(x_i; \theta))\right)\right]$;
      end for
      $w_i^{\text{unn}} = \texttt{sigmoid}(-\gamma(\texttt{PM}_i - \beta))$;
    end for
    $\omega_i = M \times w_i^{\text{unn}} / \sum_j w_j^{\text{unn}}, \forall i \in [m]$; ▷ $\omega_i = 1$ during burn-in period.
    $\theta \leftarrow \theta - \eta \nabla_\theta \sum_i \left(\beta \omega_i \texttt{KL}(\mathbf{p}(x_i + \delta_i^{(T)}; \theta) || \mathbf{p}(x_i; \theta)) - \sum_i \log \mathbf{p}_{y_i}(x_i; \theta)\right)$;
  end for
end for

Probabilistic Margins for Instance Reweighting in Adversarial Training

Algorithm:

Algorithm 4 MAIL-MART: The Overall Algorithm.

Input: a network model with the parameters θ; and a training dataset S of size n.
Output: a robust model with parameters θ^*.

for $e = 1$ **to** `num_epoch` **do**
  for $b = 1$ **to** `num_batch` **do**
    sample a mini-batch $\{(x_i, y_i)\}_{i=1}^{m}$ from S; ▷ mini-batch of size m.
    for $i = 1$ **to** `batch_size` **do**
      $\delta_i^{(0)} = \xi$, with $\xi \sim \mathcal{U}(0, 1)$;
      for $t = 1$ **to** T **do**
        $\delta_i^{(t)} \leftarrow \texttt{Proj}\left[\delta_i^{(t-1)} + \alpha \texttt{sign}\left(\nabla_\theta - \log \mathbf{p}_{y_i}(x_i + \delta_i^{(t-1)}; \theta)\right)\right]$;
      end for
      $w_i^{\text{unn}} = \texttt{sigmoid}(-\gamma(\texttt{PM}_i - \beta))$;
    end for
    $\omega_i = M \times w_i^{\text{unn}} / \sum_j w_j^{\text{unn}}, \forall i \in [m]$; ▷ $\omega_i = 1$ during burn-in period.
    $\theta \leftarrow \theta - \eta \nabla_\theta \left(-\sum_{i=1}^{m} \omega_i \texttt{BCE}(x_i + \delta_i^{(T)}, y_i; \theta) + \beta \texttt{MKL}(x_i, \delta_i^{(T)}; \theta)\right)$;
  end for
end for

Our Approach

Dynamic PGD with Momentum: Dynamic PGD updates the perturbation δ over iterations to generate adversarial examples, incorporating momentum for improved effectiveness.

- Initialize δ as a zero vector.
- Set momentum parameter β.
- Choose step size α.
- Set maximum perturbation limit ε.

Perturbation Update Here is the update process: For each iteration t:

1. Compute the gradient of the loss: $\nabla X\, L(f(X + \delta; \theta), y)$.
2. Update the momentum: $m(t) = \beta \cdot m(t-1) + \nabla X\, L(f(X + \delta; \theta), y)$.
3. Determine the direction of the update: $step = sign(m(t))$.
4. Apply the update: $\delta(t) = clip(\delta(t-1) + \alpha \cdot step, -\varepsilon, \varepsilon)$.

Our Approach

Feature Importance Adversary Attacks

1. Calculating Feature Importance: We use the gradient of the model's output with respect to the input to identify each feature's importance.

2. Applying Perturbations: Perturbations are applied more significantly to features with higher importance, creating effective adversarial examples.

Given a model f and a loss function L, feature importance I is calculated as:

$I = |\nabla X\, L(f(X; \theta), y)|$

Perturbations δ are then applied based on Inorm = I/ max(I), using:

$$\delta^{(t)} = \text{clip}\left(\delta^{(t-1)} + \alpha \cdot \text{sign}\left(\nabla_X L(f(X + \delta^{(t-1)}; \theta), y)\right) \cdot I_{\text{norm}}, -\epsilon, \epsilon\right)$$

Challenges and Timeline

- Our attack model may introduce noticeable changes to actual/natural image.
- Finetuning attack factor to modify attack invariant features.

Apr 6th to 12th
Understanding and Implementing Previous work

Apr 13th to 22nd
Implementing our ideas, training, and testing

Apr 23rd to 27th
Report writing

Evaluation

- Datasets: CIFAR10, CIFAR100
- Validation with previous work: we check the results of new adversarial attacks and will compare the model's performance with new attacks when compared with old ones.

Note: We use PGD and Dynamic PGD on MAIL defense and see the results. Also, will include a method called FIA.

References

1. Towards Defending against Adversarial Examples via Attack-Invariant Features
2. Improving Adversarial robustness requires revisiting misclassified examples
3. Probabilistic Margins for Instance Reweighting in Adversarial Training

Q&A

/ Feedback