CSCD94 Week 4 Review

Dominik Luszczynski

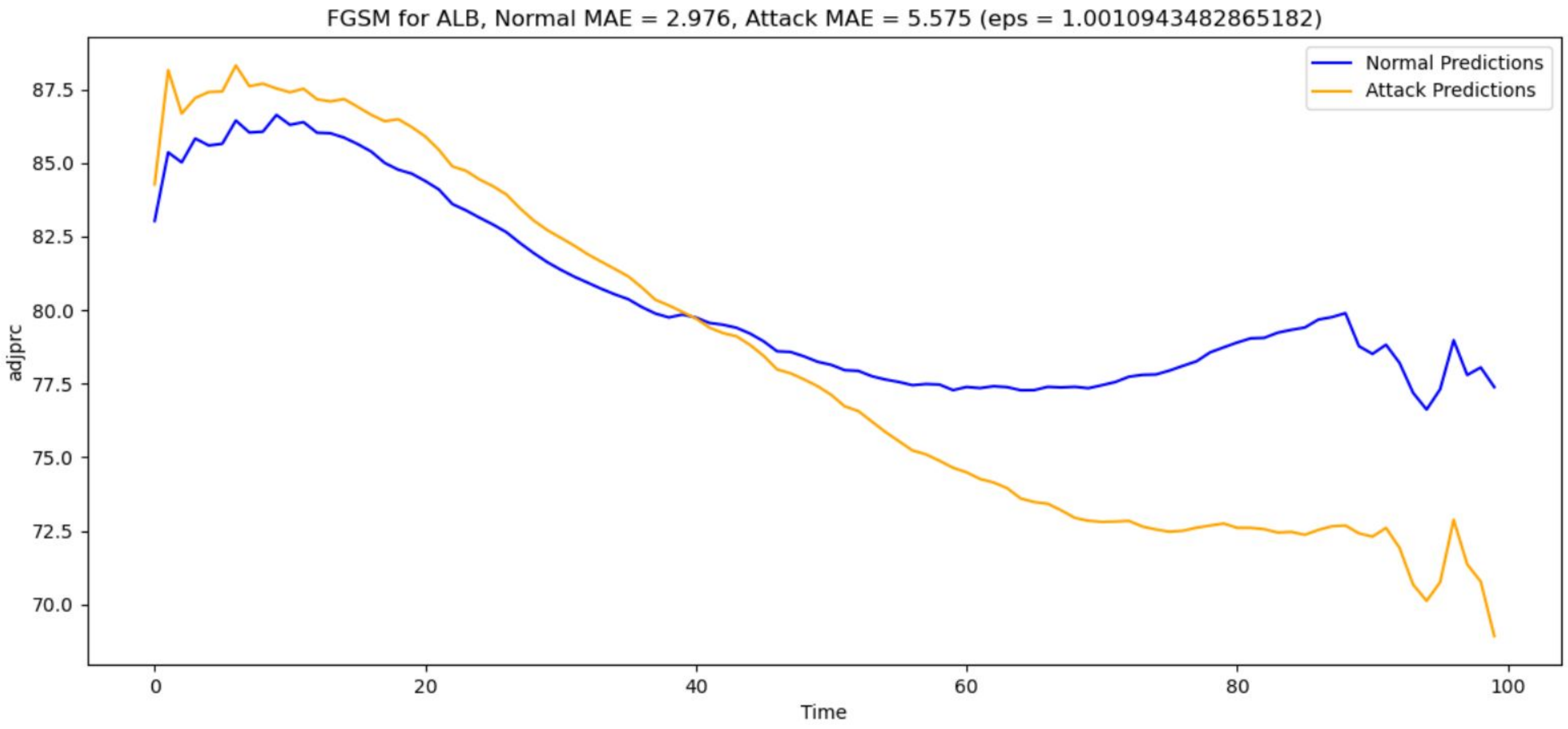
Last Week: Fast Gradient Sign Method

$$adv_x = x + \varepsilon * sign(\nabla_x J(\theta, x, y)$$ [1]

Steps of attack [2]:

1) Call model(payload) with the normal adjprc to get predictions
2) In my case, take the average prediction for each time step of the 0.5 quantile (used scatter_add to preserve gradients)
3) Compute the loss function (whatever loss you want - I chose MAE)
4) Compute the gradient (model.zero_grad() -> loss.backward() -> grad = adjprc.grad.data)
5) New attack_adjprc = adjprc + eps * sign(grad), where eps=0.5

FGSM

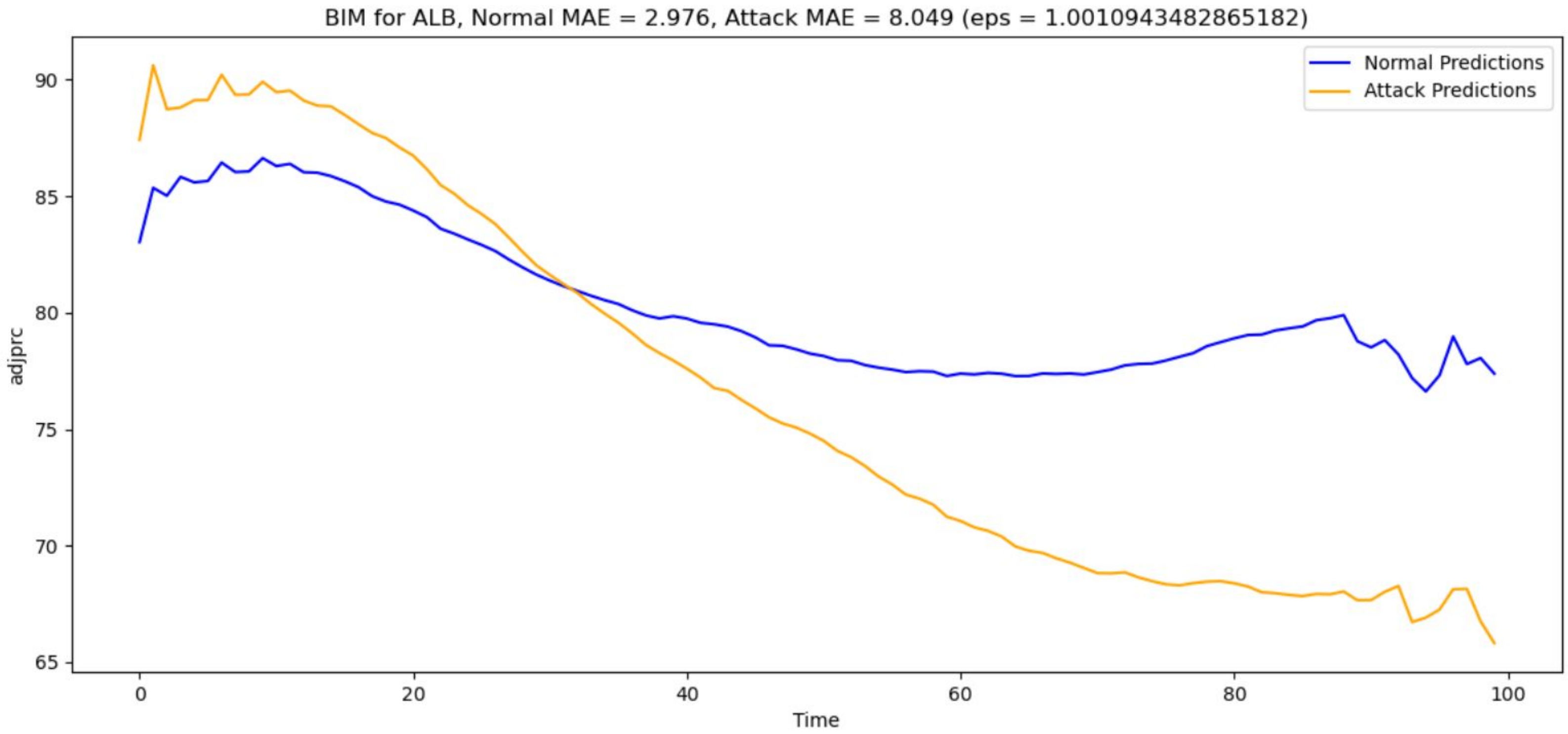

FGSM for ALB, Normal MAE = 2.976, Attack MAE = 5.575 (eps = 1.0010943482865182)
Normal Predictions
Attack Predictions
adjprc
Time
87.5
85.0
82.5
80.0
77.5
75.0
72.5
70.0
0
20
40
60
80
100

Basic Iterative Method [3]

1) Initialize adv = original adjprc
2) Loop through the number of set num_iterations (hyperparameter)
    a) Ensure adv has gradients active
    b) Call model(adv)
    c) Compute the loss(adjprc, predictions)
    d) with torch.no_grad()
        i) adv = adv + step_size * sign(loss.grad)
        ii) Clip adv to be between adjprc - epsilon and adjprc + epsilon
    e) adv.detach() (fresh computational graph)

Note that FGSM is a special case of BIM where num_iterations = 1 and step_size = 1

BIM Example

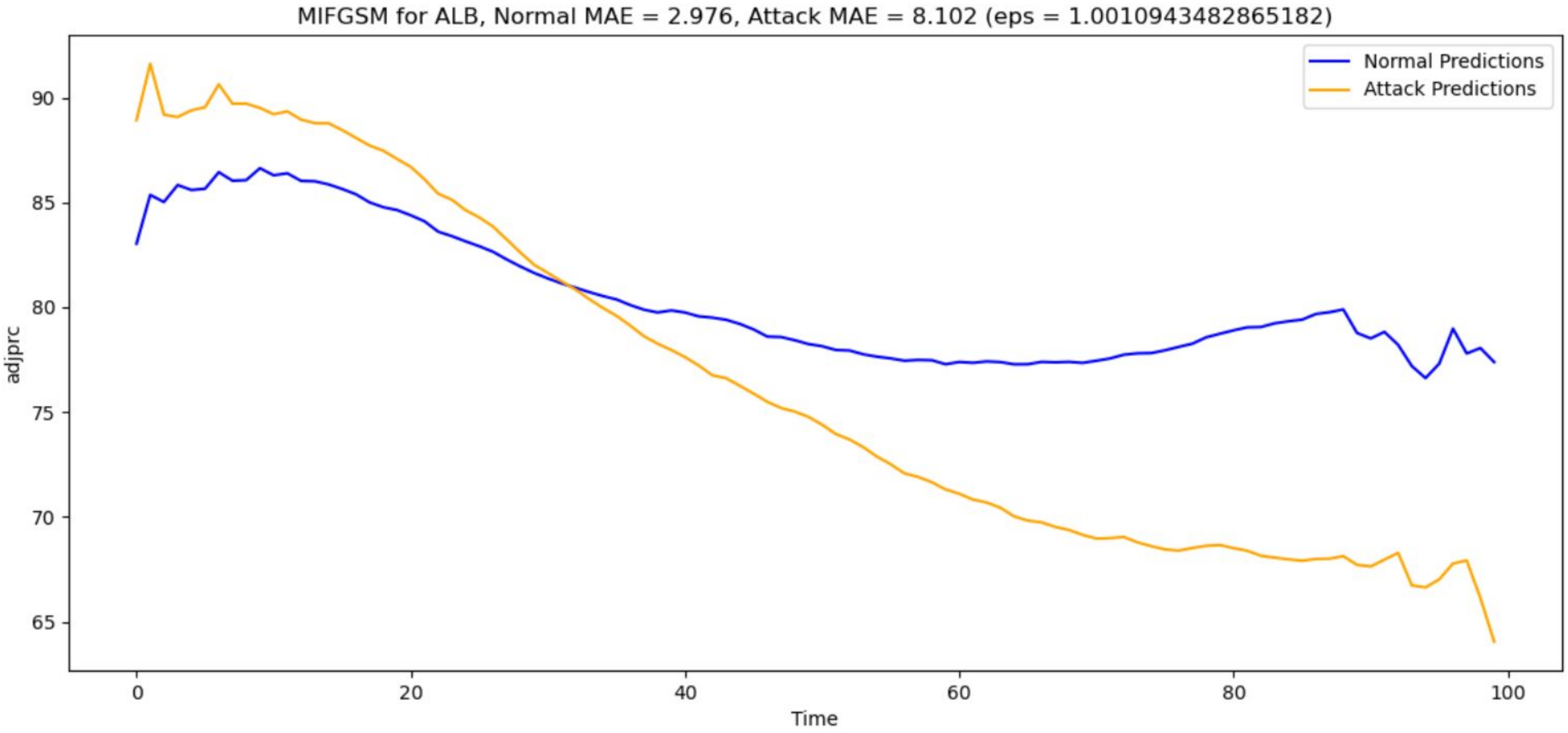


BIM for ALB, Normal MAE = 2.976, Attack MAE = 8.049 (eps = 1.0010943482865182)
Normal Predictions
Attack Predictions
adjprc
Time
90
85
80
75
70
65
0
20
40
60
80
100

Momentum Iterative FGSM [4]

1) Initialize adv = original adjprc and g = 0
2) Loop through the number of set num_iterations (hyperparameter)
   a) Ensure adv has gradients active
   b) Call model(adv)
   c) Compute the loss(adjprc, predictions)
   d) with torch.no_grad()
      i) g = decay * g + loss.grad / 1-norm(loss.grad)
      ii) adv = adv + step_size * sign(g)
      iii) Clip adv to be between adjprc - epsilon and adjprc + epsilon
   e) adv.detach() (fresh computational graph)

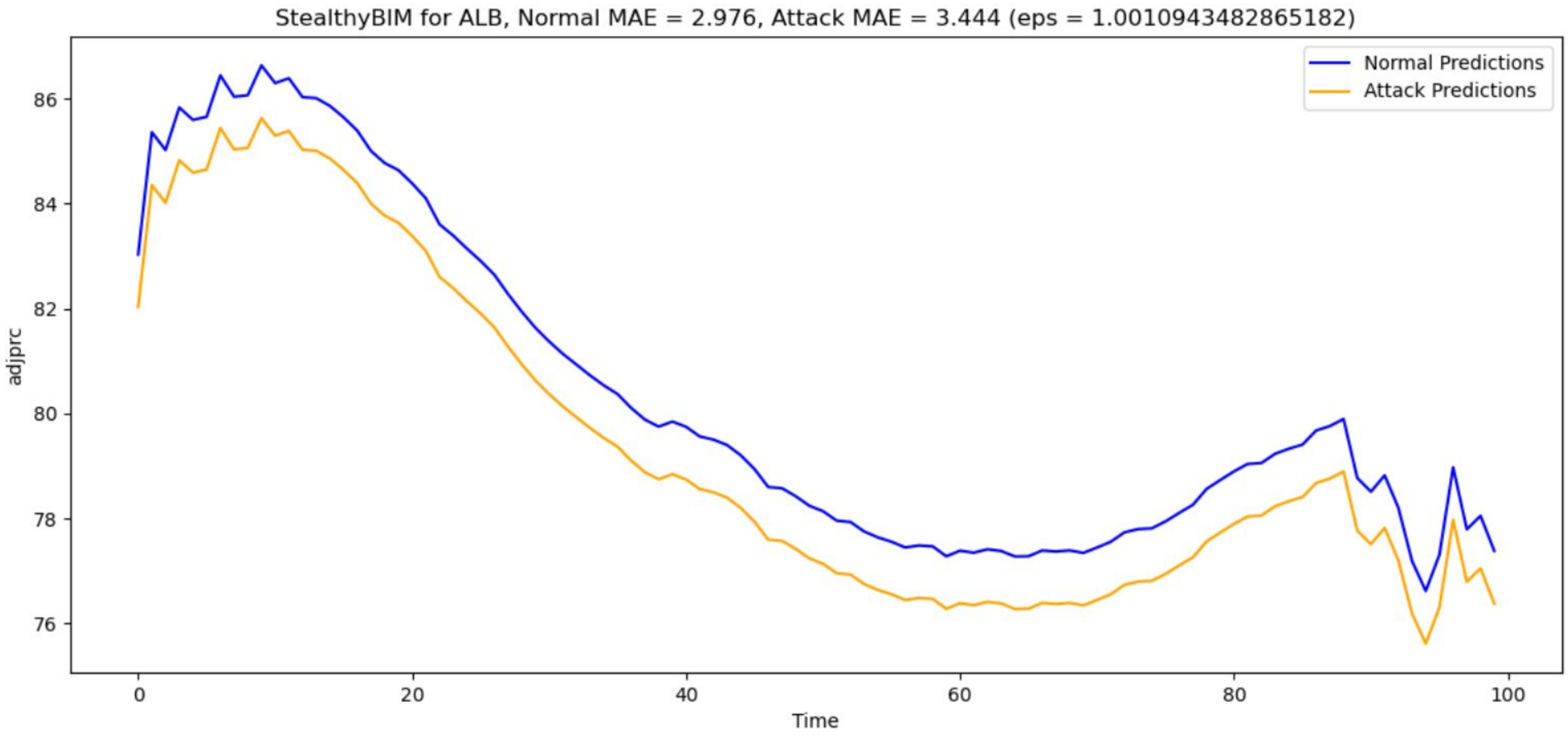
MI-FGSM EXAMPLE



“Stealthy” BIM [5]

1) Initialize adv = original adjprc
2) Loop through the number of set num_iterations (hyperparameter)
   a) Ensure adv has gradients active
   b) Call model(adv)
   c) Compute the loss(adjprc, predictions)
   d) with torch.no_grad()
      i) adv = adv + step_size * sign(loss.grad)
      ii) Clip adv to be between adjprc - epsilon and adjprc + epsilon
      iii) Compute cosine similarities between adjprc and **adv, adjprc - epsilon, adjprc + epsilon**
      iv) adv = whichever is most similar
   e) adv.detach() (fresh computational graph)

Stealthy BIM Example

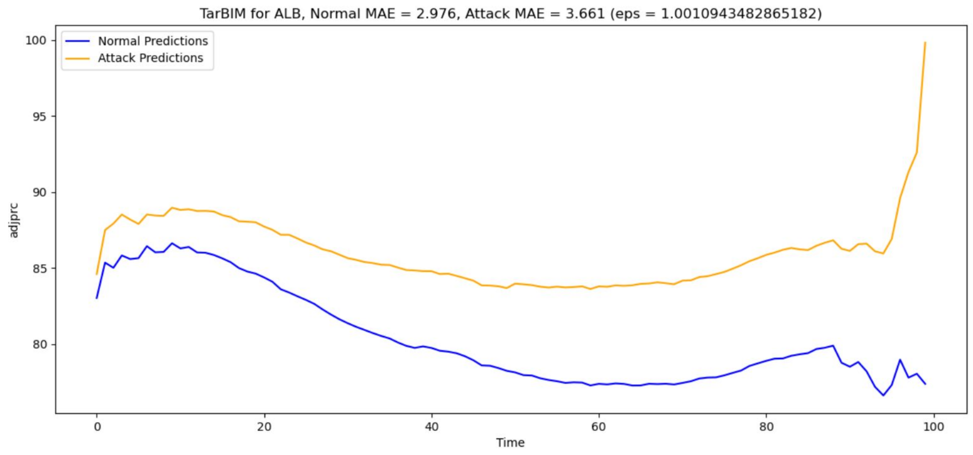


Targeted BIM [5]

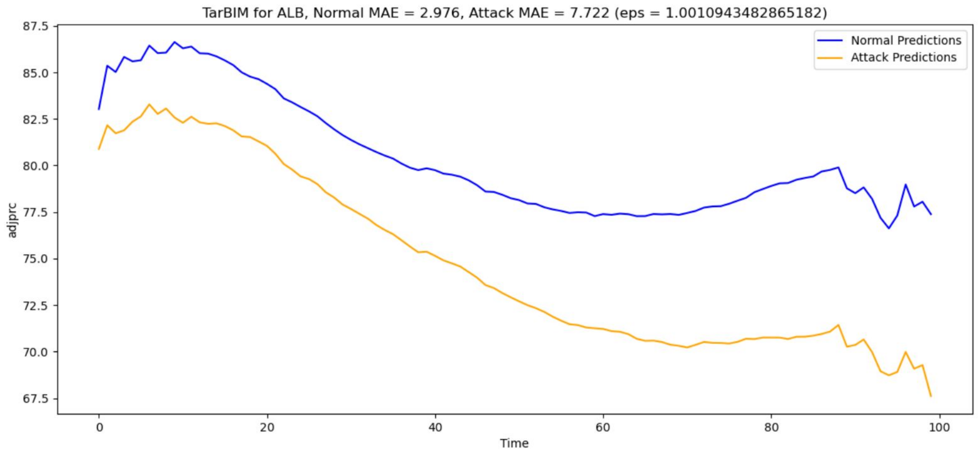
1) Initialize adv = original adjprc
2) Initialize target = adjprc + direction * margin (2 hyperparameters)
3) Loop through the number of set num_iterations (hyperparameter)
    a) Ensure adv has gradients active
    b) Call model(adv)
    c) Compute the loss(target, predictions)
    d) with torch.no_grad()
        i) adv = adv **- step_size * sign(loss.grad)**
        ii) Clip adv to be between adjprc - epsilon and adjprc + epsilon
    e) adv.detach() (fresh computational graph)

TarBIM Example

Target Above



Target Below

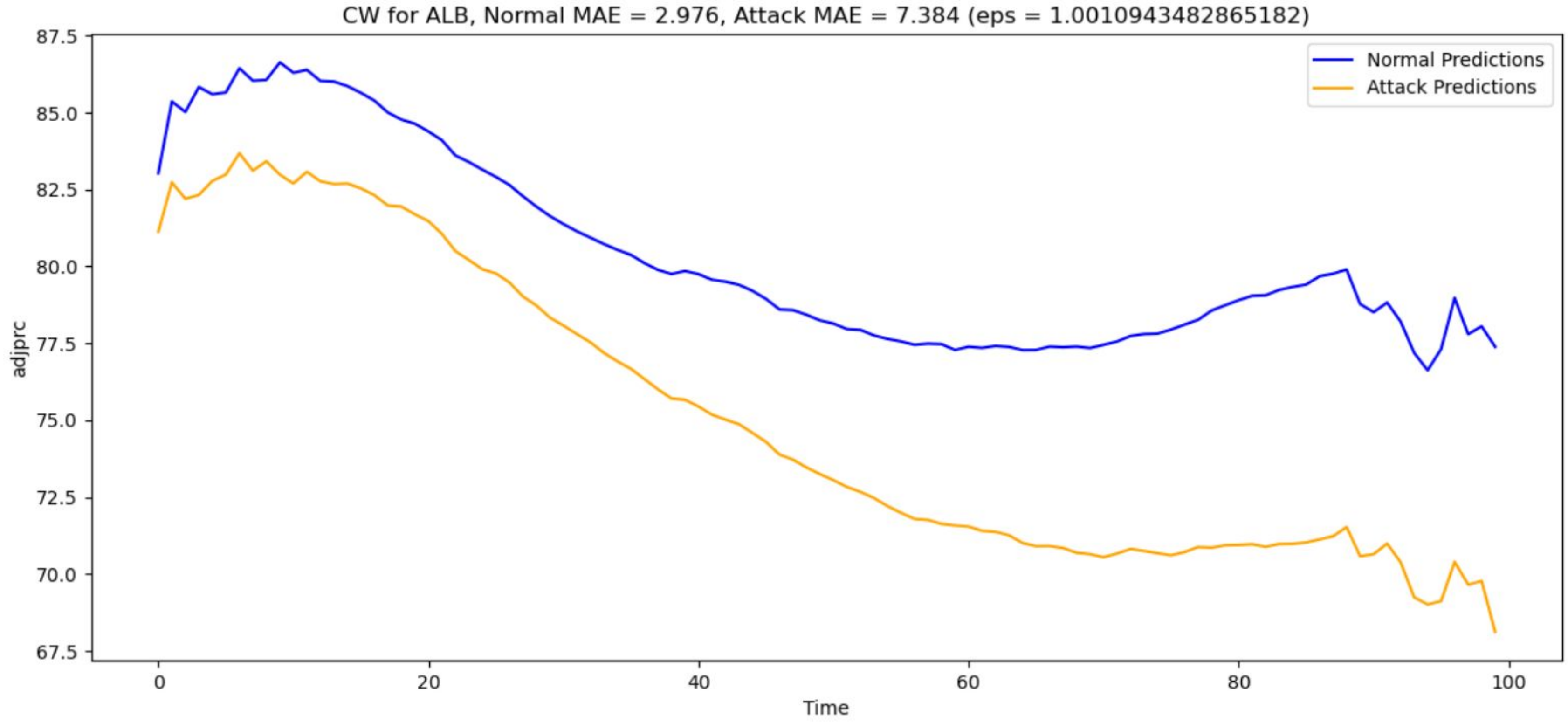


C&W Attack [6]

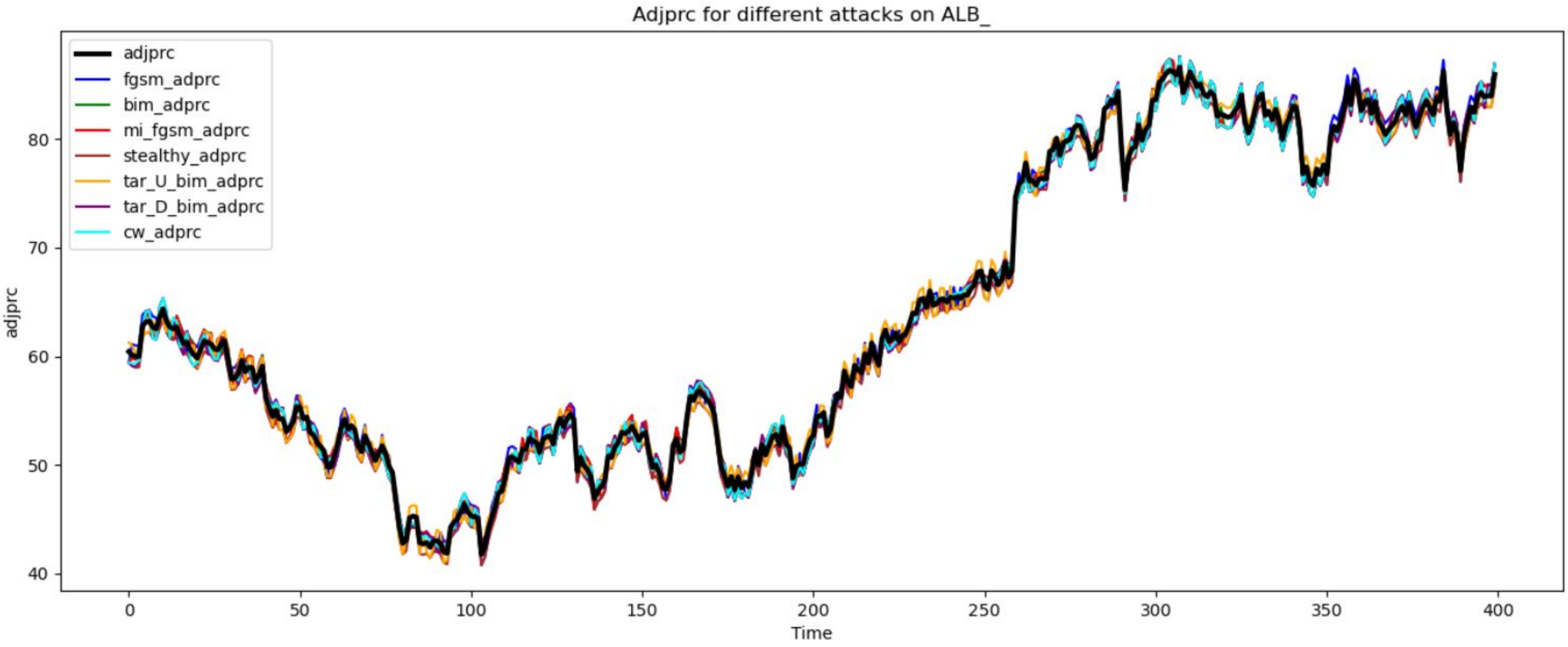
- Idea is to optimize the noise rather than iteratively improving adv

1) Initialize target (just like the Targeted BIM)
2) Initialize a noise vector *n* that requires gradients and initialize an optimizer
3) Loop through the number of set num_iterations (hyperparameter)
    a) Clamp the noise vector to be between -epsilon and epsilon
    b) adv = adv + noise
    c) Call model(adv)
    d) Loss = c * loss(pred, target) + size_penalty * ||adv||_2 (c and size_penalty are hyperparams)
    e) loss.backward()
    f) optimizer.step()

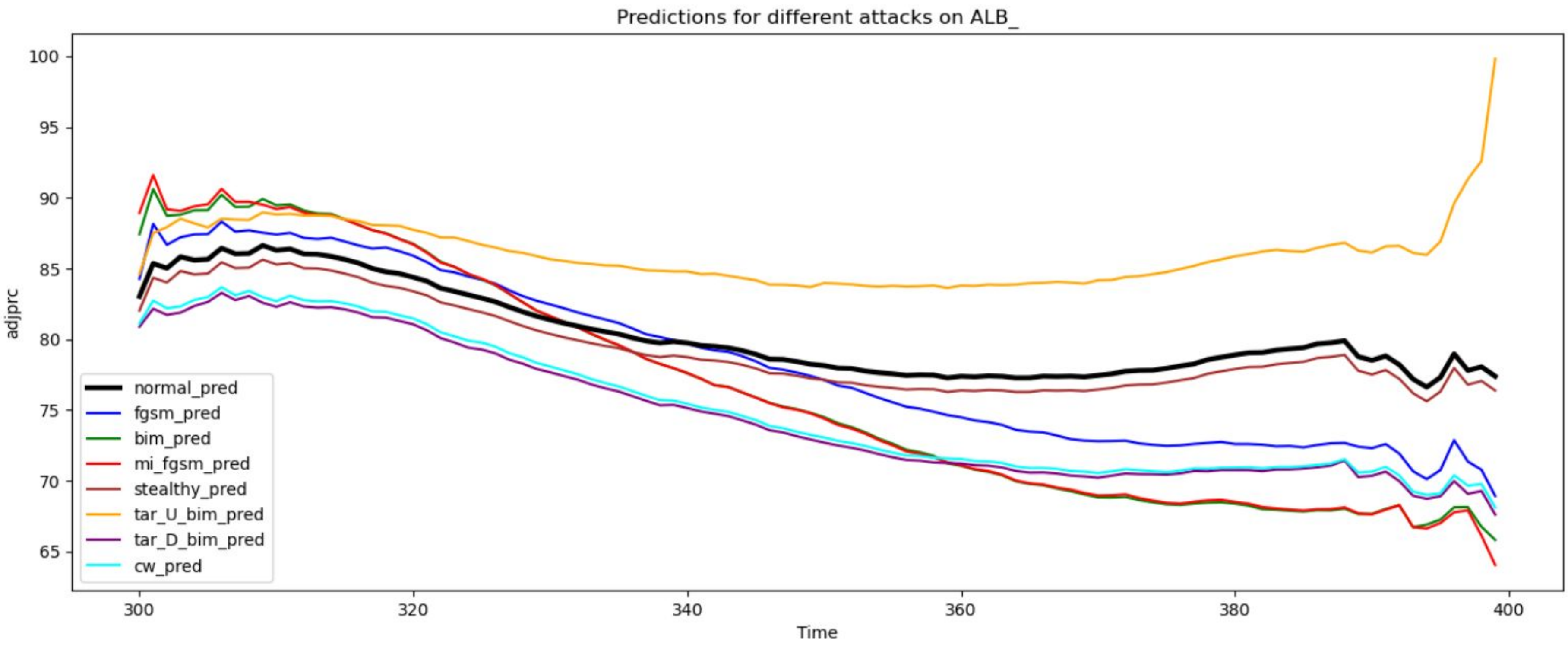
C&W Example (Target is below the normal adjprc)



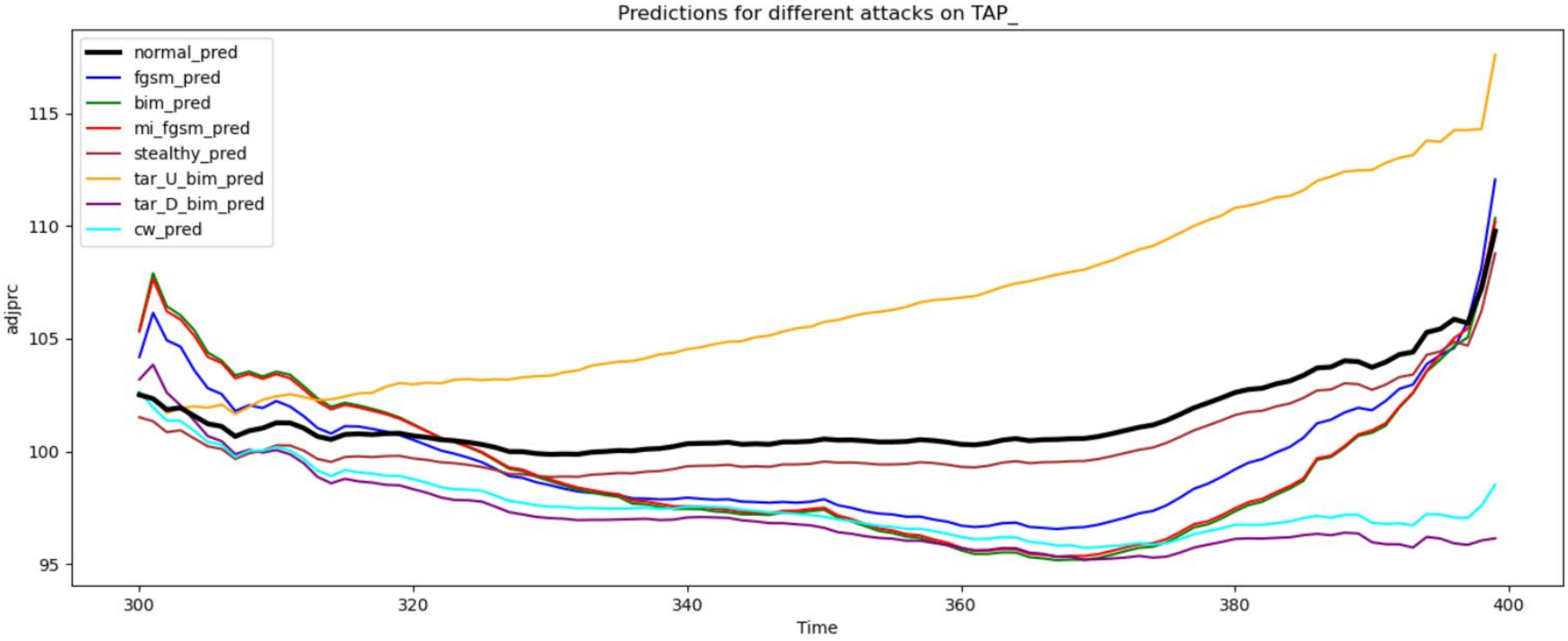
Comparison Between Attacks



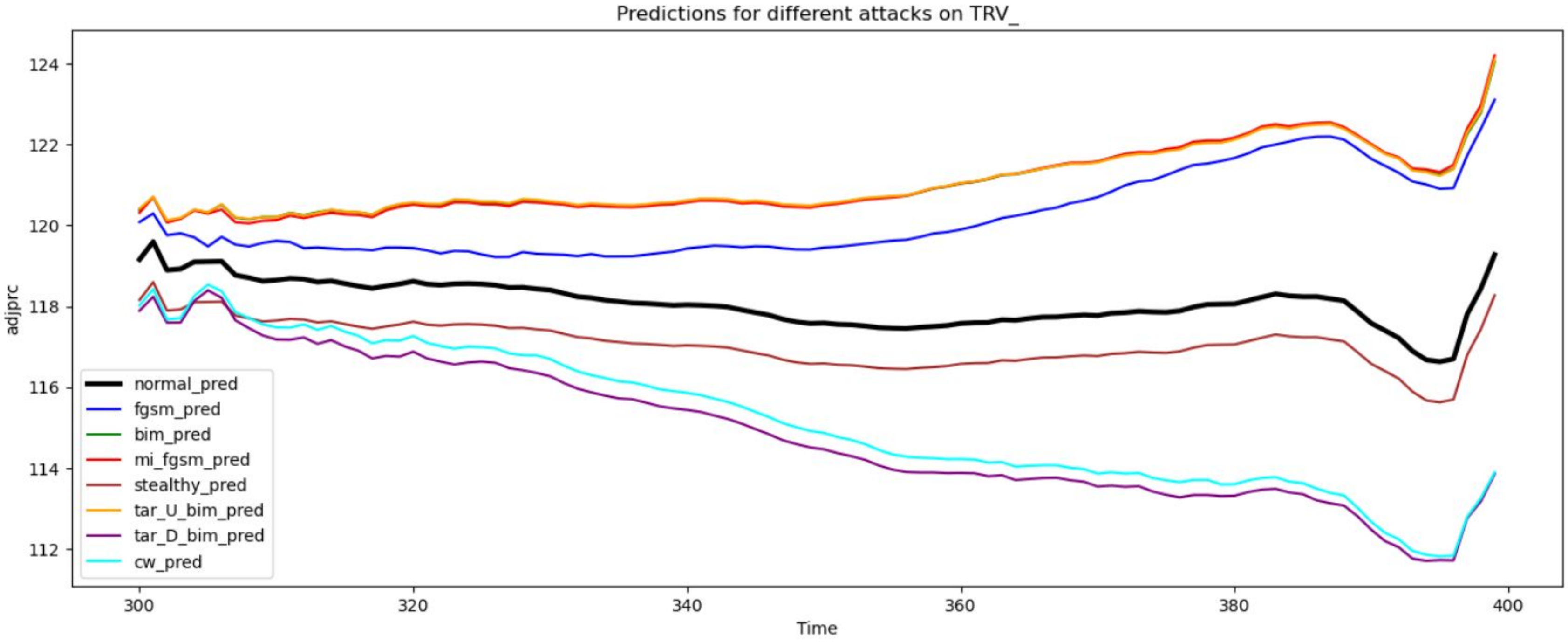
Comparison Between Attacks



Other Examples



Other Examples



Next Steps

- Experiment with different hyperparameters for the attacks.
- Start the GAN implementation

Citations

[1] J. Sen and S. Dasgupta, “Adversarial Attacks on Image Classification Models: FGSM and Patch Attacks and their Impact,” 2023, doi: 10.48550/arxiv.2307.02055.

[2] M. Gallagher, N. Pitropakis, C. Chrysoulas, P. Papadopoulos, A. Mylonas, and S. Katsikas, “Investigating machine learning attacks on financial time series models,” Computers & security, vol. 123, pp. 102933-, 2022, doi: 10.1016/j.cose.2022.102933

[3] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” 2016, doi: 10.48550/arxiv.1607.02533.

[4] Y. Dong et al., “Boosting Adversarial Attacks with Momentum,” in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, 2018, pp. 9185–9193. doi: 10.1109/CVPR.2018.00957.

[5] Z. Shen and Y. Li, “Temporal characteristics-based adversarial attacks on time series forecasting,” Expert systems with applications, vol. 264, pp. 125950-, 2025, doi: 10.1016/j.eswa.2024.125950.

[6] G. Pialla et al., “Time series adversarial attacks: an investigation of smooth perturbations and defense approaches,” International journal of data science and analytics, vol. 19, no. 1, pp. 129–139, 2025, doi: 10.1007/s41060-023-00438-0.