Problem Setup:

Train an ensemble model by enlarging the diversity among base models to improve its robustness against adversarial attacks, aiming to show that the state of the art attacks cannot attack the trained robust ensemble model to a certain extent.

Related Work/Papers:

Improving Adversarial Robustness via Promoting Ensemble Diversity:

Pang, Tianyu, et al. “Improving Adversarial Robustness via Promoting Ensemble Diversity.” *ArXiv.org*, 29 May 2019, arxiv.org/abs/1901.08846.

Towards Evaluating the Robustness of Neural Networks:

Carlini, Nicholas, and David Wagner. “Towards Evaluating the Robustness of Neural Networks.” *ArXiv.org*, 22 Mar. 2017, arxiv.org/abs/1608.04644.

Kariyappa, S. and Qureshi, M. K., “Improving Adversarial Robustness of Ensembles with Diversity Training”, arXiv:1901.09981, 2019.

Tramer, F., Carlini, N., Brendel, W., and Madry, A., “On Adaptive Attacks to Adversarial Example Defenses”, arXiv:2002.08347, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A., “Towards Deep Learning Models Resistant to Adversarial Attacks”, arXiv:1706.06083, 2017.

Yang, H., “DVERGE: Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles”, arXiv:2009.14720, 2020.

Andrew, I., Shibani, S., Dimitris, T., Logan,E., Brandon, T., and Aleksander, M., “Adversarial Examples Are Not Bugs, They Are Features”, arXiv:1905.02175, 2019

Ambra, D., Macro, M., and Maura, P., “Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks”, arXiv:1809.02861, 2018

Adversarial-attacks-pytorch:

```
@article{kim2020torchattacks,
  title={Torchattacks: A Pytorch Repository for Adversarial Attacks},
  author={Kim, Hoki},
  journal={arXiv preprint arXiv:2010.01950},
  year={2020}
}
```

XGBoost Documentation:

https://xgboost.readthedocs.io/en/latest/

Progresses:

Dataset chosen:

1. Found 2 MNIST datasets online but could not divide the dataset into estimation and validation parts.
2. Tried Sklearn.datasets. But the MNIST dataset Sklearn used are small and not regular MNIST dataset. Only (1797, 64)
3. Finally chose torchvision MNIST datasets.
    a. Training dataset:
        i. Batch size = 128
        ii. Normalized using (0.5, 0.5)
    b. Test dataset:
        i. Batch size = 1000
        ii. Normalized using (0.5, 0.5)

Parameters of XGBoost model:

```
'tree_method' : 'gpu_hist',
'max_depth': 5,              # the maximum depth of each tree
 'eta': 0.3,                 # the training step for each iteration
'silent': 1,                 # logging mode - quiet
 'objective': 'multi:softmax',   # multiclass classification using the softmax objective
'num_class': 10              # the number of classes that exist in this datset
```

Number of rounds:

500

Accuracy:

98%

Initial Model(Suppose to be a simple model because need to combine all simple models into ensemble model)**:**

Model Parameters:

```
class HNet(nn.Module):
    def __init__(self):
        super(HNet, self).__init__()
        self.flatten = nn.Flatten()
```

```
                self.fc1 = nn.Linear(784, 128)
                self.fc2 = nn.Linear(128, 64)
                self.fc3 = nn.Linear(64, 10)


        def forward(self, x):
                x = self.flatten(x)
                x = self.fc1(x)
                x = F.relu(x)
                x = self.fc2(x)
                x = F.relu(x)
                 x = F.softmax(x, dim = 1)
                return x
```

Optimizer Chose:

optimizer = optim.Adam(initial_model.parameters(), lr=0.003)

Loss Function Chose:

criterion = nn.CrossEntropyLoss()

Training Epochs:

65

Accuracy:

95.277%

Residual Model:

Model Parameters:

```
        class HNet(nn.Module):
                def __init__(self):
                        super(HNet, self).__init__()
                        self.flatten = nn.Flatten()
                        self.fc1 = nn.Linear(784, 128)
                        self.fc2 = nn.Linear(128, 64)
                        self.fc3 = nn.Linear(64, 10)


                def forward(self, x):
                        x = self.flatten(x)
                        x = self.fc1(x)
                        x = F.relu(x)
                        x = self.fc2(x)
                        x = F.relu(x)
```

x = F.softmax(x, dim = 1)
return x

Optimizer Chose:
optimizer = optim.Adam(initial_model.parameters(), lr=0.003)

Loss Function Chose:
criterion = nn.MSELoss()

Training Epochs:
40

Training Epochs ADP:
30

Number of Models Used:
3

Optimized Gamma Found:
[0.1789, -0.0662, 0.0360]

Loss for training epochs:
Submodel 1: 0.012757 ~ 0.017339
Submodel 2: 0.019255 ~ 0.021827
Submodel 3: 0.020141 ~ 0.022674

Loss for ADP training epochs:
Submodel 1: -4.436037
Submodel 2: 1.290421
Submodel 3: 7.013631

Optimized Gamma Found ADP:
[0.0122, -0.0125, -0.0010]

Ensemble Accuracy:
95.282%

Ensemble Accuracy ADP:
95.277%

Applying Carlini-Wagner Attack:
On the single model:

Robust Accuracy: 42.52%

On the ensemble model:

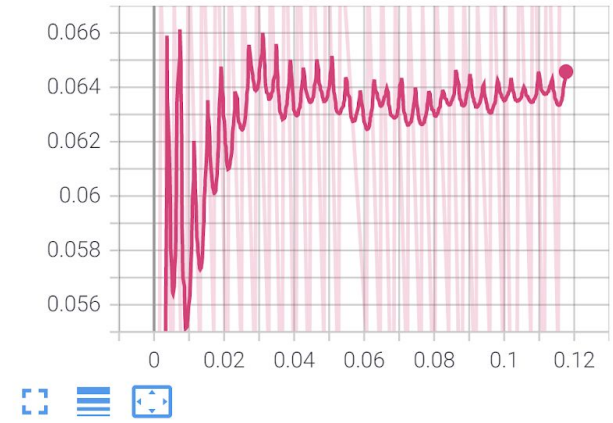
Robust Accuracy: 44.53%

On the ADP ensemble model:

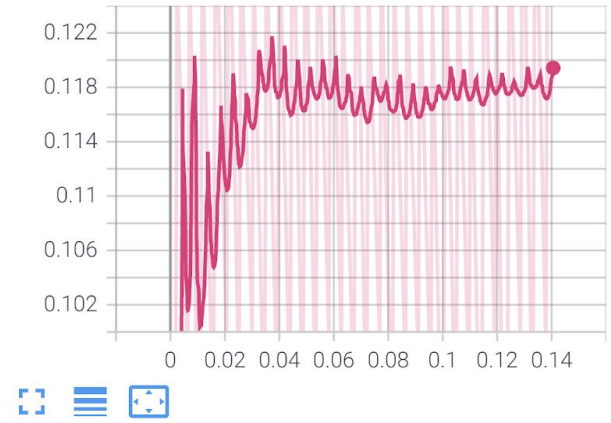
Robust Accuracy: 42.19 %

Graphs of CE_LOSS, EE_LOSS, DET_LOSS and LOG_DET:

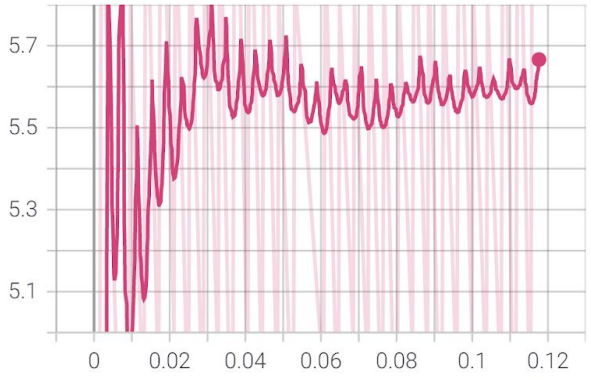
ce_loss_1
tag: train/ce_loss_1



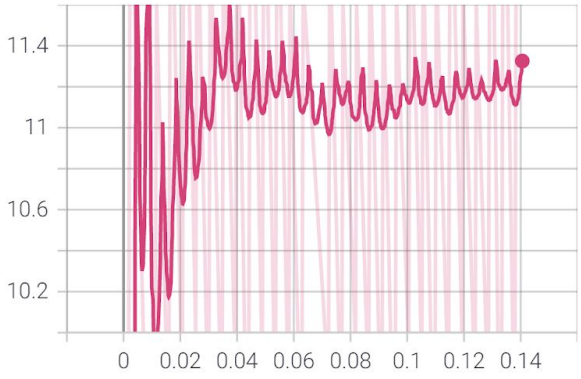
ce_loss_2
tag: train/ce_loss_2

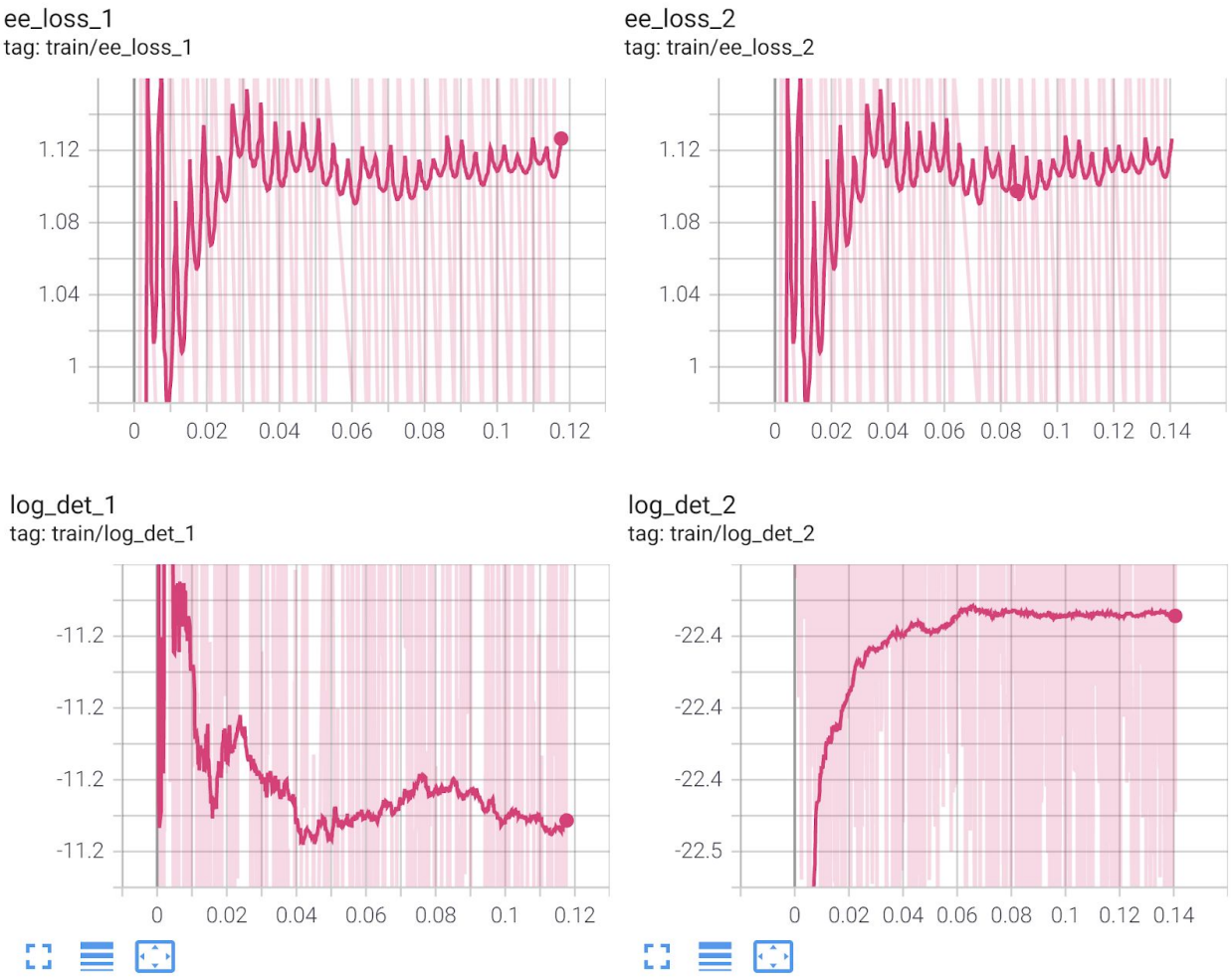


det_loss_1
tag: train/det_loss_1



det_loss_2
tag: train/det_loss_2





Results:

1. **XGB model did a really good job on training MNIST dataset but failed to attack because we cannot get its gradient**
2. **Self implemented Gradient boosting model: Increased the accuracy from 95.277% to 95.282%, robust accuracy from 42.52% to 44.53%**
3. **Applied Adaptive Diversity Promoting strategy on training gradient boosting ensemble models: The accuracy barely changes. Although the determinant of the volume matrix did increase, which implies that the diversity is enlarged, but ADP strategy failed to perform a better job than the original gradient boosting model. The robust accuracy drops from 42.52% to 42.19%.**

Problems met:

1. I read attack.py in torchattacks but cannot understand why should we divide the attack modes into three different modes: 1. Original, 2. Targeted 3. Least likely: https://github.com/Harry24k/adversarial-attacks-pytorch/blob/master/README.md

   Explanation:
   Targeted: create a new y` - some noise
   Original: predict labels other than y
   Least likely: confidence on each class and choose the lowest confidence.

2. How to implement CW attack and then apply the attack to XGB model. Cannot directly use the CW attack in the torchattacks package, because in XGB model I do not have a parameter called training.

   Solution:
   Transform into pytorch model.

3. Found package called hummingbird.ml which has a convert function can convert both sklearn models and XGB models. However, hummingbird only supports XGBClassifier, XGBRanker, XGBRegressor. Therefore I cannot transform XGB tree boost into pytorch model.

   Solution:
   Implement own pytorch ensemble model using the same boosting strategy: Gradient Boosting

4. When multiple models are trained in one time, pytorch failed to find the previous loss.

   Solution:
   Set Loss function to retain_graph = True to keep track of every loss calculated

5. Cannot optimize a constant value using optimizer function, strange errors
   Solution:
   Use torch.ones to create tensor for the constant and put the tensor into a dictionary when applying optimizer

6. The optimizer runs, but the value of the gamma never changes which were always 1.

   Solution:
   Change the tensor of gamma into random

7. Since I have to set retain_graph = True, my GPU memory is insufficient for training the Residual Models too many times or training too many models

Solution:
Limited the models trained to 3 and the epochs each Residual Model used to 3

8. The torchattacks CW attack can only attack 1 model at a time

Solution:
Created own CW attack by modifying the structure of torchattacks CW attack, so that the CW_ensemble_attack can attack multiple models as a whole in one time.

.
AdaBoost was the first really successful boosting algorithm developed for binary classification, while Xgboost uses gradient boosting decision trees as its algorithm. AdaBoost can be viewed as a special case with a particular loss function. In order to achieve more flexibility, Xgb has to use gradient boosting algorithm.

Loss Function AdaBoost has to optimize:

$$\min_{\alpha_n,\beta_n} L\left(y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)\right)$$