Use of Generative Adversarial Neural Network for Cosmic ray simulations

Pranav Sampathkumar

April 28, 2021

Wasserstein Distance (GANPlify): Fail

Architecture

Generator: $[1000 : 256, ELU] > [256 : 256, ELU, Dropout]x7 > [256 : d]$
Discriminator:
$[Embedding, Aggregate] > [256 : 256, LeakyRLU, Dropout]x7 > [256 : 1, Sigmoid]$
Embedding:
$[Conv1d : 256, K = 1, S = 1, Pad = 0]x2 > [Conv1d : 32, K = 1, S = 1, Pad = 0]$
Aggregate: Standard deviation of embedded pairwise distance

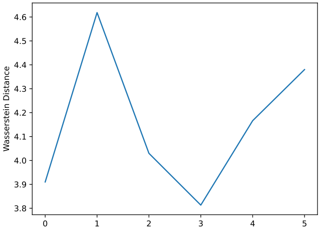


Figure: Wasserstein Distance

Distribution (GANPlify): Fail

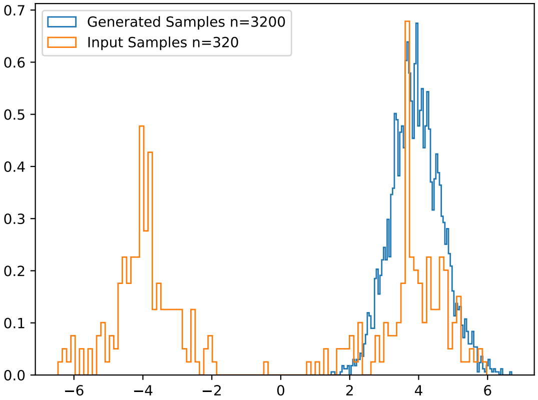


Figure: Generated Distribution Sample

Wasserstein Distance (Vannila): Success

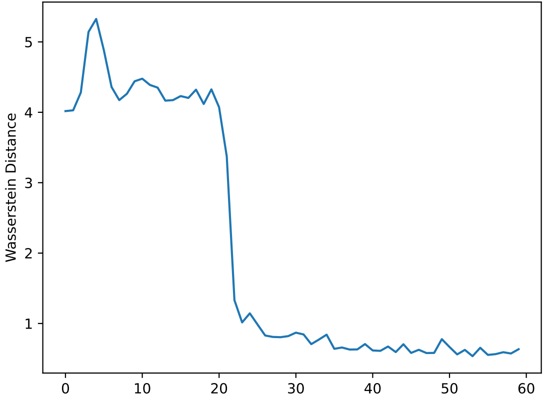
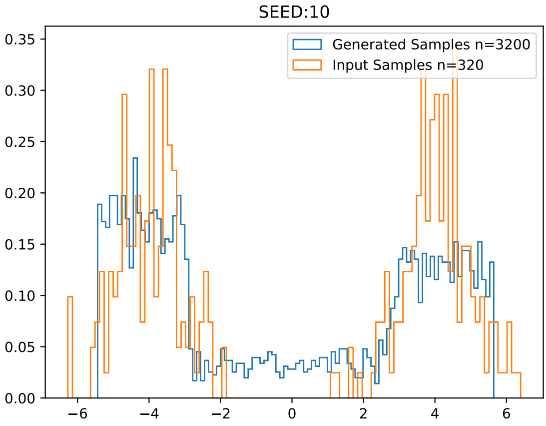


Figure: Wasserstein Distance

Distribution (Vannila): Success

Architecture

Generator: $[1:64:32:1, LeakyReLU]$
Discriminator: $[1:64:32, LeakyReLU] > [32:1, Sigmoid]$



Distribution (Vannila): Fail

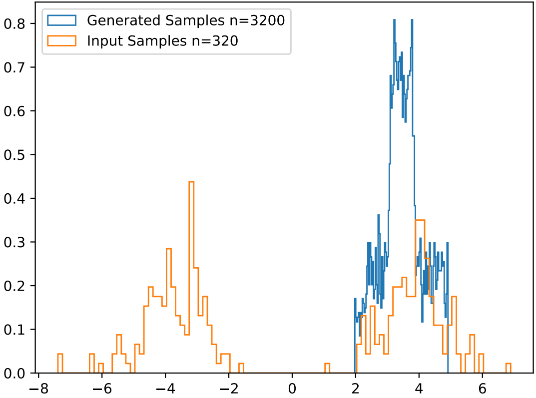


Figure: Generated Distribution Sample

Wasserstein Distance (Vannila): Fail

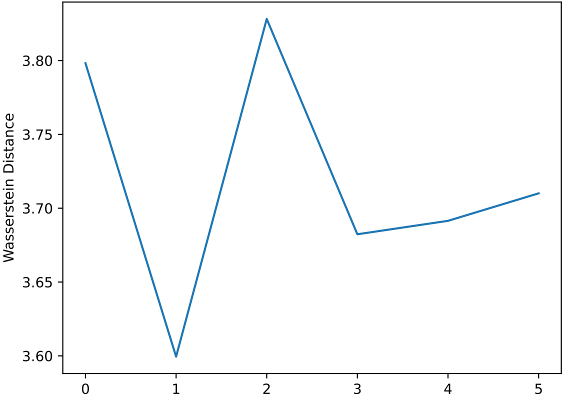


Figure: Wasserstein Distance

Learning Strategy

For each Epoch:

- Train Discriminator
  - Compute loss for real values
  - Freeze Generator
  - Generate fake values
  - Compute loss for fake values
  - Take a minimization step towards lower average loss: $(\text{real} + \text{fake})/2$
- Train Generator
  - Generate fake values
  - Pass it to discriminator
  - Minimize the difference between discriminator prediction and “all real”

  Note that in this step, discriminator isn’t frozen and is also backpropagated

Things done

- Modularize the code to separate models from training strategies and incorporate strategy as a discrete parameter
- 2 Models one based on Vannila GAN and one based on GANPlify
- Make the code reproducible (Almost)
- Implement Wasserstein GAN (Almost)

Things to do

- vectorized rejection sampling in python
- Implement Bayseian hyperparameter optimization with optuna
- High dimensional plotting with HiPlot