Neural Network-Based Neutrino Interaction Simulator

Author[1*]

[1]Department, Institution, City, State or Province, Postal Code, Country

Correspondence
Author, Department, Institution, City, State or Province, Postal Code, Country
Email: correspondingauthor@email.com

Present address
[†]Department, Institution, City, State or Province, Postal Code, Country

Funding information
Funder, Funder Department, Grant/Award Number:

The Neutrino Physics (NP) and Deep Learning (DL) communities share the same ambition: building mathematical models that best fit the observation and produce principled predictions of explainable outcomes. Meanwhile, they struggle against similar obstacles: a sophisticated collection of uncertainties resulting from different sources. Limited knowledge of neutrino detectors' simulation and interaction cross-sections give rise to the most significant systematic uncertainties in precision neutrino measurements, which barricades building a neutrino mass model. Simultaneously, today's deep neural network models are usually unable to understand their uncertainty, e.g., epistemic and aleatoric uncertainty, mainly resulting from a lack of data reconstruction from the high-level representation. These uncertainties have made the current deep learning algorithms are prone to adversarial errors. Therefore, we propose the ν-GAN project, which aims to constrain these uncertainties by integrating data-driven deep learning and physics-based models. Instead of merely using image-like neutrino interaction eventdisplays to train a deep learning model, the physics-informed deep learning (PIDL) solution employs observed experimental data in conjunction with the known standard model of particle physics to achieve more accurate simulation of subatomic particle interactions and propagation through matter. The proposed simulator not only aims to help physicists to better understand neutrino interactions but also aids in stepping stone toward quantification uncertainties from deep neural networks (DNN). Additionally, the project offers the capability of online learning to establish an integrated research-education-outreach program that will: (i) transform the fundamental understanding of deep learning grounded with neutrino physics knowledge; (ii) promote participation by undergraduates, in particular, underrepresented groups; (iii) inspire teenagers to pursue STEM-related careers.

Abbreviations: High Energy Physics (HEP); Generative Adversarial Network (GAN);

[*]Equally contributing authors:

1 | INTRODUCTION

Deep learning has found its way into neutrino physics research, which is moving into the precision measurement era, in particle classification [1] [2] [3] [4] and interaction reconstruction [5] [6]. Deep neural networks, trained with simulated datasets, have proven to work better than traditional algorithms in the above two fields that deep learning technologies are introduced to broader applications. Nevertheless, the beauty and clearness of the trained DNN models, which assert neutrino interactions category and energy, is at present obscured by two clouds. I. The first came into existence with the neutrino-nucleus interactions and detector simulation, and was dealt with Monte Carlo generators, such as Geant4 [7] and GENIE [8]; it involved the question, how could we accurately simulate the neutrino interactions, e.g., 2p2h (multi-nucleon) effects on charged-current neutrino-nucleus reactions? II. The second is the long-neglected uncertainty issue, which comes from different deep neural network architectures.

Cloud I: *Limited knowledge of neutrino detectors' response and interaction cross-sections.*
Since Ganchang Wang first proposed to detect neutrinos by beta capture in 1942 [9], neutrino detection has become well-established [10] [11] [12], and experimental neutrino physics is moving into the precision measurement era [13] [14] [15]. These neutrino experiments' success would not have been possible without extensive Monte Carlo simulations. Most of the contemporary neutrino experiments' simulation chain consists of a set of modules executed in sequence. Regularly, it starts with a neutrino-nucleus interactions generator that provides the final states which observed as primary particles in various detectors. The next module is detector simulation, which describes the detector geometry and the passage of the generated particles through the detectors' materials and magnetic field. The final module then simulates the readout method, including how the detector electronics calibrate and process the measured signals. The NuMI[1] Off-axis ν_e Appearance (NOvA), an accelerator-based neutrino experiment as shown in Fig 1 & Fig 2, applies the described simulation chain (GENIE and Geant4) to produce its simulation data sets [16] [17].

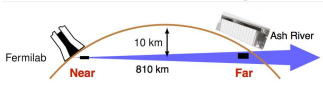

FIGURE 1 NOvA is an off-axis long-baseline neutrino experiment with two functionally identical detectors, studying oscillations over the 810km baseline and the world's most powerful ν_μ (NuMI) beam [18]. Image source NOvA

FIGURE 2 The energy spectrum of the neutrinos produced by the NuMI beam is measured by the Near Detector (ND) located 1km away from the NuMI target. The neutrinos are subsequently detected 810 km away in the Far Detector(FD) near Ash River, MN. The FD is 14.6 mrad off-axis so that the resulting narrow neutrino-energy spectrum peaks around 2 GeV, near the first oscillation maximum [19]. Image source Alex Sousa

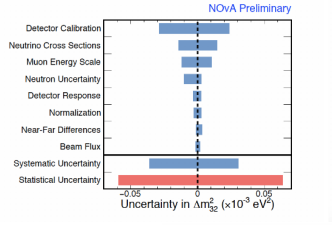

FIGURE 3 With the increasing total exposure, the statistical uncertainty will dramatically decrease. The systematic uncertainties, mainly caused by the detector and neutrino interaction simulation, will barricade NOvA to precisely measure the neutrino mass splitting (Δm_{32}). [20] [21]. Image source Patricia Vahle

Yet, there is no complete theory from first principles to handle the difficult task of predicting interactions with the complex nuclear environment. Consequently, interaction generators employed various effective field theories to describes the interactions and subsequent states. Hence, the hadronic hard-scattering processes at different momentum scales

[1] The Neutrinos at the Main Injector beam

can be characterized by a relatively more uncomplicated nuclear dynamic model. This approach has been sufficient for past neutrino experiments, yet increasing precision measurement in modern neutrino experiments has begun to reveal cracks in the foundation [20] [21]. Additionally, detector simulation uncertainty, mainly resulting from the calibration and response process, have a significant impact on the potential reach of neutrino experiments, Fig 3 represented their influence on NOvA mass splitting measurement.

Cloud II: *Epistemic and aleatoric uncertainty in deep learning.*

Deep learning has come to a long hard road since its early beginnings. The broad availability and affordability of powerful computing devices have enabled the rapid development of sophisticated neural network architectures like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Generative Adversarial Networks (GAN). Physicists, in the high energy physics community, have come up with spectacular applications in fields like particle classification and interaction energy regression, where deep neural networks (DNN) have proven to perform as good as or even better than traditional dedicated identification and reconstruction algorithms, which have been developed for processing experiment data over decades. These models, trained based on simulated data set, are often taken blindly and assumed to be accurate, unfortunately, which is not always the case and may bring disastrous consequences. Today's DNN architectures are usually unable to understand/explain their uncertainty[22] [23]. DL researchers tragically encountered the first fatality from an assisted driving system in 2016 [24]. The DNN-based autopilot did not notice

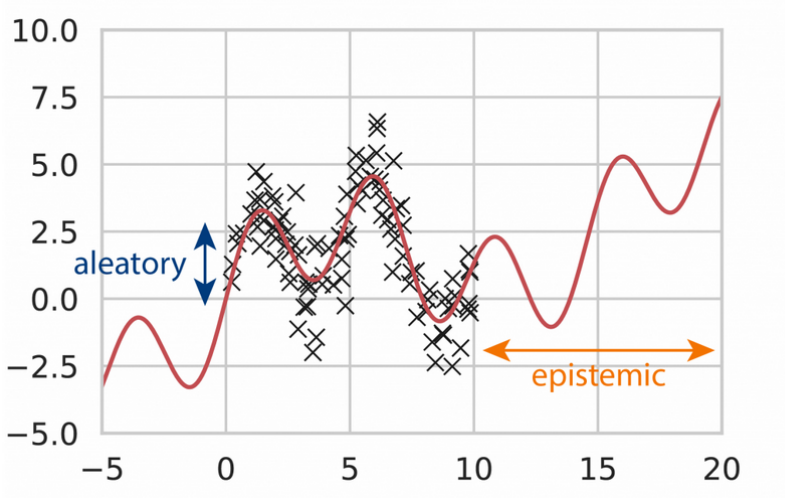

FIGURE 4 We take a simple data model as an example to explain the epistemic and aleatoric uncertainty. It includes one input dimension and one target dimension. The black data points are backgrounds that are generated based on a deterministic mean function, the red plot. The vertical spread for similar data values can be considered aleatory uncertainty. The lack of data on parts of the input domain will result in epistemic uncertainty due to the mean function is not known in practical scenarios. Credit by Simon Bachstein

the white side of the tractor, so the brake was not applied. If the trained model was able to assign a high level of uncertainty to its mistaken predictions, the autopilot system might have been able to make better decisions and likely avoid tragedy [25]. The primary sources of deep learning uncertainty are epistemic and aleatoric uncertainty, as described in Fig 4. Epistemic uncertainty, also called model uncertainty, captures our ignorance about the lack of training data in some regions of the input domain, e.g., the unseen inputs which are caused by the deviation between

the phenomenological model predictions and the measured data. [26]. Aleatoric uncertainty, on the other hand, catches our doubt concerning information, which our data cannot explain, such as the measured data includes detector flash [27]. The same concern arises from the neutrino physics community, as presented in Fig 5.

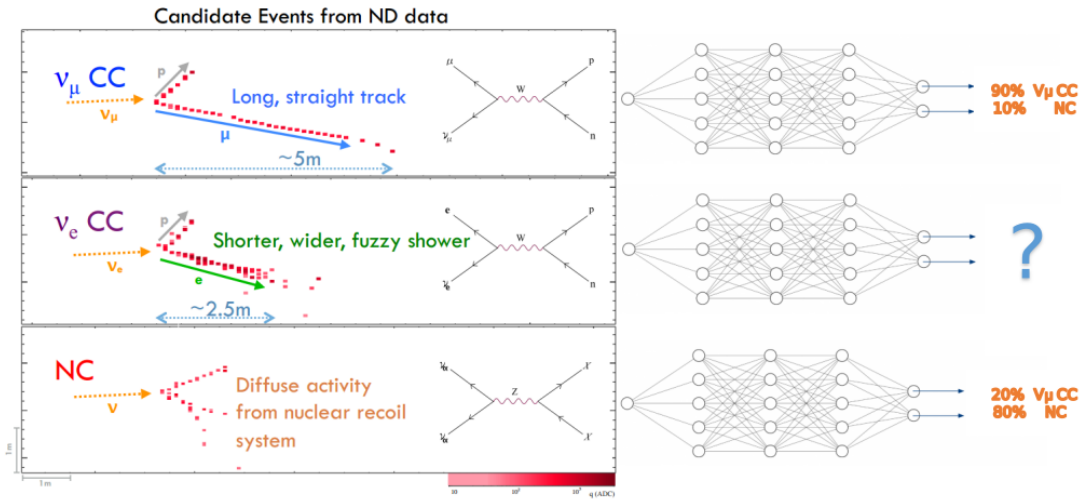

FIGURE 5 Let us take the convolutional visual network (CVN), a popular CNN-based neutrino interaction classifier, to explain the deep learning model uncertainty. The left plot shows the three types of neutrino-nuclear interactions in NOvA near detector (ND). Suppose, the CVN was trained with only two types of neutrino interactions: charged current (CC) and neutral current (NC). When the classification model, is sure about the input, it assigns a high probability to the class. If it sees both CC and NCs' features, on the other hand, it will give an indifferent result. Nevertheless, if the input is a CC, which has never seen by the model, its output will be unpredictable. Though the NOvA CVN was trained with all the three types of interactions, the model uncertainty is still an issue needed to be considered. The training input datasets are produced by the NOvA simulation chain, which employed several effective field theory models to generate the simulation interactions, which are expected to deviate from the measured data in the NOvA detectors. Image source Patricia Vahle

We propose the *ν*-GAN project, a physics-informed deep learning solution, to address these concerns from both neutrino physics and deep learning communities. We approach these problems by modeling generative adversarial networks trained with measured raw data from different neutrino experiments (in particular test beam tagged data) while employing state-of-art methods to explain, estimate and constrain the uncertainty from the used DNN architectures. By comparing our simulated results with the ones coming from the simulation chains, we can not only have the chance to understand the employed effective theory models better but also progress towards the long-term goal of deep learning assisted scientific discovery from experimental data without making prior assumptions about the system [28] [29]. Additionally, our knowledge of the underlying physics laws and the detected data provides us with the clean input datasets and organized environment to study the DNN architectures, therefore helping both communities better understand how DL operates and potentially design more robust network architectures [30] [31].

This work is organized as follows. Section 2 introduces the datasets and interprets them from a probability distribution view (where the statistics fit in), instead of just call them "image-like" eventdisplays; Section 3 describe the employed effective theory models applied by the simulation chain; Section 4 presents various generative and adversarial learning algorithms and different methods to estimate and constrain epistemic and aleatoric uncertainty from the employed architectures; Section 5 detailed introduction to the *ν*-GAN project, the goal, the methods, and the expected results;

Section 6 reports the on-going works.

2 | DATA SAMPLES

3 | NEUTRINO SIMULATION AND PHENOMENOLOGY

4 | GENERATIVE AND ADVERSARIAL LEARNING

5 | THE ν-GAN PROJECT

6 | ONGOING WORK

We report all the on-going works and results here.

6.1 | GAN Architectures Tuning

We start the project from familiar and tuning various GANs in the market, as detailed presented in the appendix C. We present the first trained GAN and the simulated results here. The code can be found in appendix B.

6.1.1 | Employed Dataset

CERN Open Data has been employed for the architectures tuning analysis. There are barriers to open HENP data sets, but doing so has already contributed to scientific progress. We take our hats off to heroes who are from these groups. I propose honoring them, who are the first to open HEP data sets, Data-Man/Data-Woman.

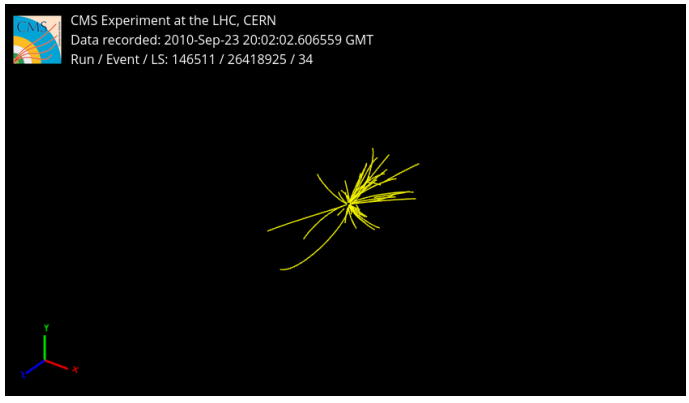

FIGURE 6 CERN Open Data: CMS Education Dataset. Data source can be found in appendix A.

6.1.2 | First GAN-based simulated interactions

We present our first GAN-based CMS interaction simulation results here.

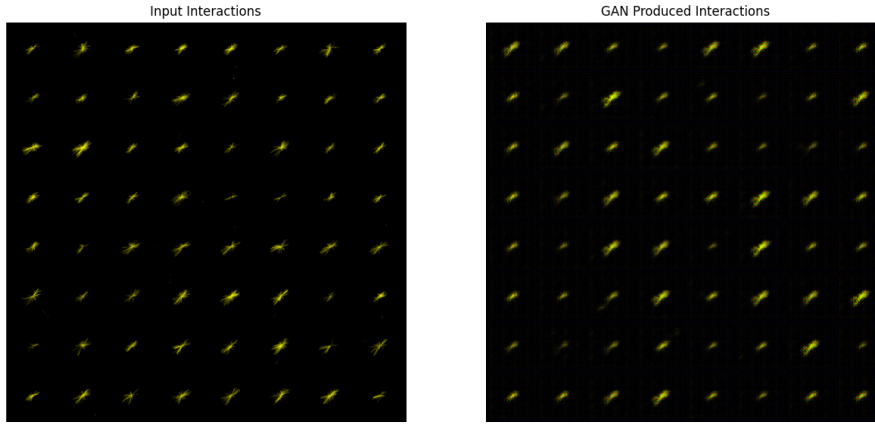

FIGURE 7 The Left ones are input data, the Right ones are GAN produced CMS interactions

Acknowledgements

Acknowledgements Here.

references

[1] Xu Y, Xu W, Meng Y, Wu B. Improving Application of Bayesian Neural Networks to Discriminate Neutrino Events from Backgrounds in Reactor Neutrino Experiments. JINST 2009;4:P01004.

[2] Aurisano A, Radovic A, Rocco D, Himmel A, Messier MD, Niner E, et al. A Convolutional Neural Network Neutrino Event Classifier. JINST 2016;11(09):P09001.

[3] Choma N, Monti F, Gerhardt L, Palczewski T, Ronaghi Z, Prabhat, et al. Graph Neural Networks for IceCube Signal Classification 2018;.

[4] Abi B, et al. Neutrino interaction classification with a convolutional neural network in the DUNE far detector 2020;.

[5] Fechner M, Abe K, Hayato Y, Iida T, Ikeda M, Kameda J, et al. Kinematic reconstruction of atmospheric neutrino events in a large water Cherenkov detector with proton identification. Phys Rev D 2009 Jun;79:112010. https://link.aps.org/doi/10.1103/PhysRevD.79.112010.

[6] Aiello S, et al. Event reconstruction for KM3NeT/ORCA using convolutional neural networks 2020 4;.

[7] Agostinelli S, Allison J, Amako K, Apostolakis J, Araujo H, Arce P, et al. GEANT4—a simulation toolkit. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 2002 11;506:250–303.

[8] Andreopoulos C, Barry C, Dytman S, Gallagher H, Golan T, Hatcher R, et al. The GENIE Neutrino Monte Carlo Generator: Physics and User Manual 2015 10;.

[9] Wang KC. A Suggestion on the Detection of the Neutrino. Phys Rev 1942 Jan;61:97–97. https://link.aps.org/doi/10.1103/PhysRev.61.97.

[10] Cowan CL, Reines F, Harrison FB, Kruse HW, McGuire AD. Detection of the free neutrino: A Confirmation. Science 1956;124:103–104.

[11] Guo X, et al. A Precision measurement of the neutrino mixing angle θ_{13} using reactor antineutrinos at Daya-Bay 2007 1;.

[12] Feldman GJ. Long-baseline accelerator neutrino experiments. In: International Conference on History of the Neutrino: 1930-2018; 2019. .

[13] Aharmim B, Ahmed SN, Anthony AE, Beier EW, Bellerive A, Bergevin M, et al. Electron energy spectra, fluxes, and day-night asymmetries of ^{8}B solar neutrinos from measurements with NaCl dissolved in the heavy-water detector at the Sudbury Neutrino Observatory. Phys Rev C 2005 Nov;72:055502. https://link.aps.org/doi/10.1103/PhysRevC.72.055502.

[14] Araki T, Eguchi K, Enomoto S, Furuno K, Ichimura K, Ikeda H, et al. Measurement of Neutrino Oscillation with KamLAND: Evidence of Spectral Distortion. Phys Rev Lett 2005 Mar;94:081801. https://link.aps.org/doi/10.1103/PhysRevLett.94.081801.

[15] Stuttard T. Neutrino oscillations and PMNS unitarity with IceCube/DeepCore and the IceCube Upgrade. PoS 2020;NuFact2019:099.

[16] Adamson P, et al. The NuMI Neutrino Beam. Nucl Instrum Meth A 2016;806:279–306.

[17] Aurisano A, Backhouse C, Hatcher R, Mayer N, Musser J, Patterson R, et al. The NOvA simulation chain. J Phys Conf Ser 2015;664(7):072002.

[18] Ayres DS, et al. The NOvA Technical Design Report 2007 10;.

[19] Adamson P, et al. Search for active-sterile neutrino mixing using neutral-current interactions in NOvA. Phys Rev D 2017;96(7):072006.

[20] Wolcott J. Impact of cross section uncertainties on NOνA oscillation analyses. PoS 2019;NuFACT2018:098.

[21] Acero MA, et al. Adjusting Neutrino Interaction Models and Evaluating Uncertainties using NOvA Near Detector Data 2020 6;.

[22] Bykov K, Hohne MMC, Muller KR, Nakajima S, Kloft M. How Much Can I Trust You? - Quantifying Uncertainties in Explaining Neural Networks. ArXiv 2020;abs/2006.09000.

[23] Antoran J, Allingham J, Hernández-Lobato JM. Depth Uncertainty in Neural Networks. ArXiv 2020;abs/2006.08437.

[24] Mukherjee S, Awadallah AH. Uncertainty-aware Self-training for Text Classification with Few Labels; 2020. .

[25] Arnez F, Espinoza H, Radermacher A, Terrier F. A Comparison of Uncertainty Estimation Approaches in Deep Learning Components for Autonomous Vehicle Applications. ArXiv 2020;abs/2006.15172.

[26] Kendall A, Gal Y. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? CoRR 2017;abs/1703.04977. http://arxiv.org/abs/1703.04977.

[27] Kendall A, Gal Y, Cipolla R. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. CoRR 2017;abs/1705.07115. http://arxiv.org/abs/1705.07115.

[28] Iten R, Metger T, Wilming H, del Rio L, Renner R. Discovering Physical Concepts with Neural Networks. Phys Rev Lett 2020 Jan;124:010508. https://link.aps.org/doi/10.1103/PhysRevLett.124.010508.

[29] Raissi M, Perdikaris P, Karniadakis GE. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. arXiv preprint arXiv:171110561 2017;.

[30] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics 2019;378:686–707.

[31] Raissi M, Perdikaris P, Karniadakis GE. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. arXiv preprint arXiv:171110566 2017;.

[32] Radford A, Metz L, Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. CoRR 2015;abs/1511.06434.

Appendices

A | EMPLOYED TRAINING DATA SET

We employed CMS Open Data for the architecture tuning. The data set also can be download from our github.

B | DEEP CONVOLUTIONAL GAN (DCGAN) SAMPLE

The presented results, Fig 7, are based on the following code, which is modified from PyTorch. The code also can be download from our github page [32]. We present the code here to detailed explain the meaning of every single line of the code to help the potential user have a better understanding of how the GANs work. (More explanation line will be added soon.)

```
from __future__ import print_function
import argparse
import os
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

dataroot = "../CMS/Education"

# Number of workers for dataloader
workers = 3
# Batch size during training
batch_size = 64
# Spatial size of training images. All images will be resized to this size using a transformer.
image_size = 64
# Number of channels in the training images. For color images this is 3
nc = 3
# Size of z latent vector (i.e. size of generator input)
nz = 100
# Size of feature maps in generator
ngf = 32
# Size of feature maps in discriminator
```

```
ndf = 32
# Number of training epochs
num_epochs = 100
# Learning rate for optimizers
lr = 0.005
# Beta1 hyperparam for Adam optimizers
beta1 = 0.5
# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1

# Create the dataset
dataset = dset.ImageFolder(root=dataroot,
                           transform=transforms.Compose([
                               transforms.Resize(image_size),
                               transforms.CenterCrop(image_size),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))
# Create the dataloader
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=workers)

# Decide which device we want to run on
device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")

# Plot some training images
real_batch = next(iter(dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:64], padding=2, normalize=True).cpu(),(1,2,0)))

# Custom weights initialization called on netG and netD
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)

# Generator Code
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
```

```
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d( ngf * 2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        return self.main(input)

# Create the generator
netG = Generator(ngpu).to(device)

# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netG = nn.DataParallel(netG, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights to mean=0, stdev=0.2.
netG.apply(weights_init)

# Print the model
print(netG)

class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
```

```
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)

# Create the Discriminator
netD = Discriminator(ngpu).to(device)
# Handle multi-gpu if desired
if (device.type == 'cuda') and (ngpu > 1):
    netD = nn.DataParallel(netD, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights to mean=0, stdev=0.2.
netD.apply(weights_init)
# Print the model
print(netD)
# Initialize BCELoss function
criterion = nn.BCELoss()
# Create batch of latent vectors that we will use to visualize the progression of the generator
fixed_noise = torch.randn(64, nz, 1, 1, device=device)
# Establish convention for real and fake labels during training
real_label = 1
fake_label = 0
# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))

### Training Loop ###
# Lists to keep track of progress
img_list = []
G_losses = []
D_losses = []
iters = 0

print("Starting Training Loop")
```

```
# For each epoch
for epoch in range(num_epochs):
    # For each batch in the dataloader
    for i, data in enumerate(dataloader, 0):
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        # Train with all-real batch
        netD.zero_grad()
        # Format batch
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, device=device)
        # Forward pass real batch through D
        output = netD(real_cpu).view(-1)
        # Calculate loss on all-real batch
        errD_real = criterion(output, label)
        # Calculate gradients for D in backward pass
        errD_real.backward()
        D_x = output.mean().item()

        # Train with all-fake batch
        # Generate batch of latent vectors
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        # Generate fake image batch with G
        fake = netG(noise)
        label.fill_(fake_label)
        # Classify all fake batch with D
        output = netD(fake.detach()).view(-1)
        # Calculate D's loss on the all-fake batch
        errD_fake = criterion(output, label)
        # Calculate the gradients for this batch
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        # Add the gradients from the all-real and all-fake batches
        errD = errD_real + errD_fake
        # Update D
        optimizerD.step()

        # (2) Update G network: maximize log(D(G(z)))
        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        # Since we just updated D, perform another forward pass of all-fake batch through D
        output = netD(fake).view(-1)
        # Calculate G's loss based on this output
        errG = criterion(output, label)
        # Calculate gradients for G
        errG.backward()
        D_G_z2 = output.mean().item()
```

```
        # Update G
        optimizerG.step()

        # Output training stats
        if i % 50 == 0:
            print('[%d/%d][%d/%d]\tLoss_D: %.4f\tLoss_G: %.4f\tD(x): %.4f\tD(G(z)): %.4f / %.4f'
                  % (epoch, num_epochs, i, len(dataloader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))

        # Save Losses for plotting later
        G_losses.append(errG.item())
        D_losses.append(errD.item())

        # Check how the generator is doing by saving G's output on fixed_noise
        if (iters % 500 == 0) or ((epoch == num_epochs-1) and (i == len(dataloader)-1)):
            with torch.no_grad():
                fake = netG(fixed_noise).detach().cpu()
            img_list.append(vutils.make_grid(fake, padding=2, normalize=True))

        iters += 1

plt.figure(figsize=(10, 5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()

fig = plt.figure(figsize=(8,8))
plt.axis("off")
ims = [[plt.imshow(np.transpose(i,(1,2,0)), animated=True)] for i in img_list]
ani = animation.ArtistAnimation(fig, ims, interval=1000, repeat_delay=1000, blit=True)
HTML(ani.to_jshtml())

# Grab a batch of real images from the dataloader
real_batch = next(iter(dataloader))

# Plot the real images
plt.figure(figsize=(15,15))

plt.subplot(1,2,1)
plt.axis("off")
plt.title("Input Interactions")
plt.imshow(np.transpose(vutils.make_grid(real_batch[0].to(device)[:64], padding=5, normalize=True).cpu(),(1,2,0)))
```

```
# Plot the fake images from the last epoch
plt.subplot(1,2,2)
plt.axis("off")
plt.title("GAN Produced Interactions")
plt.imshow(np.transpose(img_list[-1],(1,2,0)))
plt.show()
```

C | TRAINING GAN ARCHITECTURES

TABLE 1 This is a table for the tuning architectures which we can find in the market. We almost can get a new/better architecture from deep learning community, so the table will keep updating.

GAN Architecture	Released Time	Value Function	Code Source	Trained Model
DCGAN	07/0/2016			
Auxiliary Classifier GAN	20/07/2017			
Adversarial Autoencoder	31/05/2017			
BEGAN	31/05/2017			
CaloGAN	21/12/2017			