Joseph Muzzin

CSC-315

Dr. Thall

Generative Adversarial Network

In this project, I am using a GAN (generative adversarial network). An overall structure consisting of two neural networks, one called the generator and the other called the discriminator, consists of generative adversarial networks. Generator is for estimates of probability distribution and the discriminator is used to estimate the probability from sample data that comes from real data rather than being provided by the generator. The generator is trying to fool the discriminator and feed it random data as well as real data, now the discriminator will take this data and determine whether the data is true or false.

I use the PyTorch library, for the actively developed framework and use Matplotlib to work with plotting data. After importing the PyTorch library, I import “nn” to help set up the neural network in a less verbose way. I set up a random generator seed to where I can replicate the data identically on any machine. This is expressed by using the code “torch.manual_seed(111)”. The 111 number represents the random seed used to initialize the generator of random numbers, which is used to initialize the weights of the neural network. It must have the same outcomes, considering the random nature of the experiment, as long as the same seed is used.

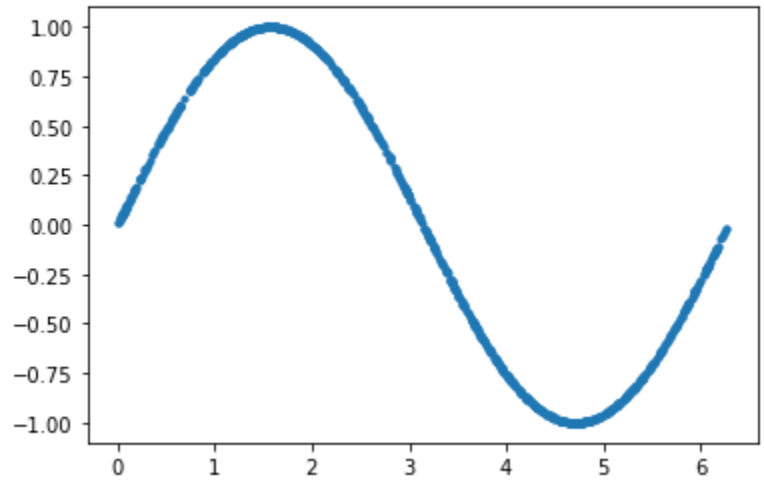
The training data consists of pairs (x1, x2) so that x2 consists of the sine value of x1 for x1 in the 0 to 2π interval. In the training set there are 1024 pairs, then initializing the train_data. After initializing the train_data, store random values from 0 to 2π. After the training data, we use a PyTorch data loader. In the program this data loader is our “train_loader”, basically staggering data from the training set and returning batches of 32 samples. These samples are used to actually train the neural network.

TrainingData Code:

```
train_data_length = 1024
train_data = torch.zeros((train_data_length, 2))
train_data[:, 0] = 2 * math.pi * torch.rand(train_data_length)
train_data[:, 1] = torch.sin(train_data[:, 0])
train_labels = torch.zeros(train_data_length)
train_set = [
    (train_data[i], train_labels[i]) for i in range(train_data_length)
]
```

Plot of TrainingData:

```
[<matplotlib.lines.Line2D at 0x7fa39248d7b8>]
```



The Discriminator class we use _init_() to build the model, then having a super() to run the _init_() from the Discriminator class. We make 3 hidden layers composed of a different number of neurons, then using a representation of output that is a single neuron with sigmoidal activation which represents the probability. After the hidden layers, we use dropout to avoid overfitting. The Generator class is similar to the creation of the Discriminator class, but is the generative part of the neural network composed of two hidden layers with 16 and 32 neurons.

Data training for the project consisted of creating 300 epochs with a learning rate of 0.001 and using a loss_function to train the data. The loss_function supplies data to the

discriminator providing a binary output. In my code, I use the Adam algorithm to train both models. The Adam algorithm consists of making a use of the average of the second moments of the uncentered variance and calculating an exponential moving average of the gradient and the squared gradient.

Training with epochs:

```
[10] for epoch in range(num_epochs):
         for n, (real_samples, _) in enumerate(train_loader):
             # Data for training the discriminator
             real_samples_labels = torch.ones((batch_size, 1))
             latent_space_samples = torch.randn((batch_size, 2))
             generated_samples = generator(latent_space_samples)
             generated_samples_labels = torch.zeros((batch_size, 1))
             all_samples = torch.cat((real_samples, generated_samples))
             all_samples_labels = torch.cat(
                 (real_samples_labels, generated_samples_labels)
             )

             # Training the discriminator
             discriminator.zero_grad()
             output_discriminator = discriminator(all_samples)
             loss_discriminator = loss_function(
                 output_discriminator, all_samples_labels)
             loss_discriminator.backward()
             optimizer_discriminator.step()

             # Data for training the generator
             latent_space_samples = torch.randn((batch_size, 2))

             # Training the generator
             generator.zero_grad()
             generated_samples = generator(latent_space_samples)
             output_discriminator_generated = discriminator(generated_samples)
             loss_generator = loss_function(
                 output_discriminator_generated, real_samples_labels
             )
             loss_generator.backward()
             optimizer_generator.step()

             # Show loss
             if epoch % 10 == 0 and n == batch_size - 1:
                 print(f"Epoch: {epoch} Loss D.: {loss_discriminator}")
                 print(f"Epoch: {epoch} Loss G.: {loss_generator}")
```

Generated Sample:

```
latent_space_samples = torch.randn(100, 2)
generated_samples = generator(latent_space_samples)
generated_samples = generated_samples.detach()
plt.plot(generated_samples[:, 0], generated_samples[:, 1], ".")
```

```
[<matplotlib.lines.Line2D at 0x7fa391723828>]
```