# Conformalized Generative Adversarial Network

# Contents

1	Class Diagram	1
<b>2</b>	Sequence diagram	2
3	Source code	3
	3.1 Conformal GAN	3
	3.2 Synthesize data using conformal GAN	6

## 1 Class Diagram

```
conformal_gan.ConformalGAN
x_train:
p_real:
gan:
alpha:
generator:
p_fake:
discriminator:
 _init__(self, generator, discriminator, gan, x_train, alpha):
predict(self, x):
train(self, epochs, batch_size, verbose=True):
generate\_samples(self, n\_samples):
train\_with\_coverage(self, epochs, batch\_size, verbose=True, validation\_data=None):
                                         object
                      _doc__ :
                      _dict__ :
                    __slots__:
                    __module_
                      _annotations_
                    __class__(self: _T):
                    __class__(self, __type: Type[object]):
                    __init__(self):
                    \_new\_(cls: Type[\_T]):
                    __setattr__(self, name: str, value: Any):
                    __eq__(self, o: object) :
                    __ne__(self, o: object):
                    __str__(self):
                    __repr__(self):
                    __hash__(self) :
                    __format__(self, format_spec: str):
                    __getattribute__(self, name: str):
                    __delattr__(self, name: str):
                    __sizeof__(self):
                    __reduce__(self):
                    \underline{\hspace{0.3cm}} reduce\_ex\underline{\hspace{0.3cm}} (self, protocol: SupportsIndex):
                    __reduce_ex__(self, protocol: int) :
                      _dir__(self):
                      _init_subclass__(cls):
                                    typing.Hashable
                                      _hash__(self):
```

Figure 1: Class diagram for ConformalizedGAN

# 2 Sequence diagram

### ConformalizedGAN for Generator and Discriminator User ConformalGAN Generator Discriminator GAN Initialize with generator, discriminator, GAN, x\_train, alpha Train for specified epochs and batch size loop [for each sample in x] alt [p-value <= alpha] Predict for real sample Predict for generated sample Assign label based on p-value Assign label as 0 Return predictions and p-values Train with coverage for specified epochs and batch size loop [for each epoch] Train on random noise Compute coverage for training set Compute coverage for validation set Print epoch progress Return training history User ConformalGAN Generator Discriminator GAN

Figure 2: Sequence diagram for ConformalizedGAN

### 3 Source code

#### 3.1 Conformal GAN

```
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow.keras.layers import Input, Dense, Reshape,
     Flatten, Dropout, BatchNormalization, LSTM, LeakyReLU
5 from tensorflow.keras.models import Sequential, Model
6 from tensorflow.keras.optimizers import Adam
8 class ConformalGAN:
      def __init__(self, generator, discriminator, gan, x_train,
     alpha):
          self.generator = generator
          self.discriminator = discriminator
11
          self.gan = gan
          self.x_train = x_train
13
          self.alpha = alpha
14
      def predict(self, x):
          p_values = np.zeros((len(x), 2))
17
          y_pred = np.zeros((len(x),))
18
          for i, x_i in enumerate(x):
19
              # Generate a sample from the generator
              z = np.random.normal(size=(1, self.generator.
21
     input_shape[1]))
              x_gen = self.generator.predict(z)
22
              # Compute the p-value for the sample
23
              d_real = self.discriminator.predict(x_i.reshape(1,
24
     -1))[0]
              d_gen = self.discriminator.predict(x_gen.reshape(1,
     -1))[0]
              p_value = (np.sum(d_gen >= d_real) + 1) / (self.
26
     x_{train.shape}[0] + 1)
              p_values[i] = [1 - p_value, p_value]
27
              # Predict the label for the sample
              if p_value <= self.alpha:</pre>
29
                   y_pred[i] = 1
30
          return y_pred, p_values
      def train(self, epochs, batch_size, verbose=True):
32
          # Compute p-values for real and fake samples
33
          D_real = self.discriminator.predict(self.X_train)
34
          D_fake = self.discriminator.predict(self.generator.
     predict(np.random.normal(0, 1, (len(self.X_train), 100)))).
     ravel()
36
          # Compute conformal prediction intervals
          self.p_real = np.zeros_like(D_real)
38
          self.p_fake = np.zeros_like(D_fake)
39
40
          for i in range(len(D_real)):
              self.p_real[i] = np.sum(D_real >= D_real[i]) / (len(
42
```

```
D_real) + 1)
               self.p_fake[i] = np.sum(D_fake >= D_fake[i]) / (len(
43
     D_fake) + 1)
44
          # Train the GAN
          for epoch in range(epochs):
46
               # Train discriminator on real samples
47
               idx = np.random.randint(0, len(self.X_train),
48
     batch_size)
               X_real = self.X_train[idx]
49
               y_real = np.ones((batch_size, 1))
50
               d_loss_real = self.discriminator.train_on_batch(
     X_real, y_real)
52
               # Train discriminator on fake samples
53
               z = np.random.normal(0, 1, (batch_size, 100))
54
               X_fake = self.generator.predict(z)
               y_fake = np.zeros((batch_size, 1))
56
              d_loss_fake = self.discriminator.train_on_batch(
     X_fake, y_fake)
               # Train generator
59
               z = np.random.normal(0, 1, (batch_size, 100))
60
               y = np.ones((batch_size, 1))
61
               g_loss = self.gan.train_on_batch(z, y)
62
63
               # Print progress
64
               if verbose and epoch % 100 == 0:
                   print(f"Epoch {epoch}: D_loss_real={d_loss_real
66
     [0]}, D_loss_fake={d_loss_fake[0]}, G_loss={g_loss}")
67
      def generate_samples(self, n_samples):
68
          z = np.random.normal(0, 1, (n_samples, 100))
69
          samples = self.generator.predict(z)
70
          # Compute prediction intervals for samples
          p_values = np.zeros((n_samples,))
73
          for i in range(n_samples):
74
               D = self.discriminator.predict(np.expand_dims(samples
     [i], axis=0)).ravel()
               p_values[i] = np.sum(D >= D.real[i]) / (len(D) + 1)
76
          lower_bound = np.percentile(samples, (self.alpha / 2) *
     100, axis=0)
          upper_bound = np.percentile(samples, (1 - (self.alpha /
79
     2)) * 100, axis=0)
          in_prediction_interval = (p_values > self.alpha / 2) & (
80
     p_values < 1 - self.alpha / 2)</pre>
81
          return samples, lower_bound, upper_bound,
     in_prediction_interval
83
84
85
```

```
86
      def train_with_coverage(self, epochs, batch_size, verbose=
      True, validation_data=None):
           train_loss = []
87
           train_coverage = []
88
           if validation_data is not None:
               test_loss = []
90
               test_coverage = []
91
           for epoch in range(epochs):
92
               # Train the discriminator
               idx = np.random.randint(0, self.x_train.shape[0],
94
      batch_size)
               x_real = self.x_train[idx]
95
               y_real = np.ones((batch_size,))
               x_gen = self.generator.predict(np.random.normal(size
97
      =(batch_size, self.generator.input_shape[1])))
               y_gen = np.zeros((batch_size,))
98
               x = np.concatenate([x_real, x_gen], axis=0)
               y = np.concatenate([y_real, y_gen], axis=0)
100
               d_loss, d_acc = self.discriminator.train_on_batch(x,
      y)
               # Train the generator
               z = np.random.normal(size=(batch_size, self.generator
      .input_shape[1]))
               y = np.ones((batch_size,))
104
               g_loss = self.gan.train_on_batch(z, y)
               # Compute the coverage for the training set
106
               y_pred, p_values = self.predict(self.x_train)
               train_loss.append([d_loss, g_loss])
               train_coverage.append(np.mean(y_pred == y_real))
               # Compute the coverage for the validation set
               if validation_data is not None:
                   x_val, y_val = validation_data
                   y_pred_val, p_values_val = self.predict(x_val)
113
                   d_loss_val, d_acc_val = self.discriminator.
114
      evaluate(x_val, y_val, verbose=0)
                   g_loss_val = self.gan.evaluate(z, y, verbose=0)
                   test_loss.append([d_loss_val, g_loss_val])
116
                   test_coverage.append(np.mean(y_pred_val == y_val)
117
      )
               if verbose:
119
                   print("Epoch {}/{} - Discriminator Loss: {:.4f} -
120
       Discriminator Accuracy: {:.4f} - Generator Loss: {:.4f}
      Train Coverage: {:.4f}".format(epoch+1, epochs, d_loss, d_acc
      , g_loss , train_coverage[-1]))
           # Return the training history
           history = {"train_loss": train_loss, "train_coverage":
      train_coverage}
           if validation_data is not None:
123
               history["test_loss"] = test_loss
124
               history["test_coverage"] = test_coverage
               return history
```

Listing 1: Python implementation for conformalized GAN.

### 3.2 Synthesize data using conformal GAN

```
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from tensorflow.keras.layers import Input, Dense, Reshape,
5 from tensorflow.keras.models import Model, Sequential
6 from tensorflow.keras.optimizers import Adam
7 from conformal_gan import ConformalGAN
9 # Generate the training data
n_samples = 10000
seq_length = 50
12 data = np.zeros((n_samples, seq_length))
13 for i in range(n_samples):
      phase = np.random.uniform(low=0.0, high=2*np.pi)
      freq = np.random.uniform(low=0.01, high=0.1)
      amp = np.random.uniform(low=0.1, high=1.0)
      x = np.linspace(0, seq_length/freq, seq_length, endpoint=
     False)
      data[i] = amp * np.sin(2*np.pi*freq*x + phase)
20 # Normalize the data
21 data = (data - np.mean(data)) / np.std(data)
23 # Define the generator model
24 latent_dim = 10
25 generator = Sequential()
generator.add(Dense(256, input_dim=latent_dim, activation='relu')
generator.add(Dense(512, activation='relu'))
28 generator.add(Dense(1024, activation='relu'))
generator.add(Dense(seq_length, activation='tanh'))
31 # Define the discriminator model
32 discriminator = Sequential()
discriminator.add(Flatten(input_shape=(seq_length,)))
discriminator.add(Dense(512, activation='relu'))
35 discriminator.add(Dense(256, activation='relu'))
36 discriminator.add(Dense(1, activation='sigmoid'))
38 # Compile the discriminator
39 discriminator.compile(loss='binary_crossentropy', optimizer=Adam
     (0.0002, 0.5), metrics=['accuracy'])
41 # Define the GAN
42 z = Input(shape=(latent_dim,))
43 time_series = generator(z)
44 validity = discriminator(time_series)
45 gan = Model(z, validity)
47 # Compile the GAN
48 gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002,
```

```
0.5))
50 # Train the GAN with conformal prediction
alpha = 0.05
n_{train} = 8000
sa x_train_subset = data[:n_train]
54 conformal_gan = ConformalGAN(generator, discriminator, gan,
     x_train_subset, alpha=alpha)
55 train_loss, test_loss, train_coverage, test_coverage =
     conformal_gan.train_with_coverage(epochs=5000, batch_size=32,
      verbose=True)
56
57 # Generate synthetic time-series data
n_{samples} = 10
59 z = np.random.normal(size=(n_samples, latent_dim))
60 synthetic_data = generator.predict(z)
62 # Plot the training and validation loss
63 plt.plot(train_loss)
64 plt.plot(test_loss)
65 plt.title('Conformal GAN Loss')
66 plt.ylabel('Loss')
67 plt.xlabel('Epoch')
68 plt.legend(['Train', 'Validation'], loc='upper left')
69 plt.show()
71 # Plot the training and validation coverage
72 plt.plot(train_coverage)
73 plt.plot(test_coverage)
74 plt.title('Conformal GAN Coverage')
75 plt.ylabel('Coverage')
76 plt.xlabel('Epoch')
77 plt.legend(['Train', 'Validation'], loc='upper left')
78 plt.show()
80 # Plot the synthetic time-series data
81 for i in range(n_samples):
      plt.plot(synthetic_data[i])
83 plt.title('Synthetic Time-Series Data')
84 plt.xlabel('Time')
85 plt.ylabel('Value')
86 plt.show()
```

Listing 2: Python implementation for generating data using conformal GAN.