**KARADENİZ TEKNİK ÜNİVERSİTESİ**

**OF TEKNOLOJİ FAKÜLTESİ**

**YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**

**PATECTGAN**

**RAPOR**

**YUSUF ÇOLAK 404746**

**2023 -2024 BAHAR DÖNEM****İ**

**İÇİNDEKİLER**

|  |  |
| --- | --- |
|  | Sayfa No |
| İÇİNDEKİLER | II |
| ÖZET | III |
| 1. GENEL BİLGİLER | 1 |
| 1.2. Tasarladığım Patectgan | 3 |
| 2. Sözde Kod | 4 |
| 3. Algoritmanın Uygulanması | 5 |
| 4. Elde Edilen Sonuç Değerleri | 6 |

**ÖZET**

Bu proje, PATECTGAN isimli, WGAN-GP ve Özellik Eşleştirme (Feature Matching) ile güçlendirilmiş bir Generative Adversarial Network (GAN) modelinin eğitim ve değerlendirme sürecini içermektedir. Model, diyabet veri seti üzerinde eğitilerek gerçek verilerin dağılımını öğrenip yeni, benzer veriler üretmeyi amaçlamaktadır. Proje kapsamında veri ön işleme adımları, modelin eğitim süreci, kullanılan değerlendirme kriterleri ve elde edilen sonuçlar ayrıntılı bir şekilde açıklanmıştır.

**1. GENEL BİLGİLER**

**1.1. Patectgan Nedir ?**

PATECTGAN modeli, generator ve diskriminatör olmak üzere iki ana bileşenden oluşur. Generator, gürültüden gerçekçi veriler üretirken, diskriminatör, gerçek ve sahte verileri ayırt etmeye çalışır. Bu süreç, iki bileşen arasındaki rekabetçi oyun ile iteratif olarak geliştirilir.

**Giriş Verileri Nasıl Alınıyor?**

Giriş verileri, train fonksiyonu tarafından alınır ve işlenir. Bu fonksiyon, eğitim verisi olarak bir Pandas DataFrame (data), isteğe bağlı olarak kategorik kolonların isimlerini (categorical\_columns) ve ordinal kolonların isimlerini (ordinal\_columns) kabul eder. Veriler, DataTransformer sınıfı kullanılarak dönüştürülür ve böylece verilerin sayısal temsilleri elde edilir. Ardından, veriler belirli bir örnek sayısına göre bölümlere ayrılır. Bu bölümler, her bir öğretmen discriminator modeli için ayrı ayrı kullanılır. train fonksiyonunda ayrıca update\_epsilon parametresi ile gizlilik parametresi epsilon güncellenebilir. Bu bölünmüş veriler, her öğretmen discriminator'un kendi veri kümesi üzerinde eğitim almasını sağlar ve PATE mekanizması aracılığıyla diferansiyel gizlilik sağlanır.

**Generator ve Discriminator Ağları Nasıl Oluşturulmuş?**

Generator ve Discriminator ağları, PATECTGAN sınıfı içinde tanımlanır. Generator, embedding\_dim ve gen\_dim parametreleriyle belirlenen katmanlarla oluşturulur ve giriş boyutuna koşullu generator opsiyonları eklenir. Discriminator ise Discriminator sınıfında tanımlanır ve input\_dim, dis\_dims, loss ve pack parametrelerine göre bir dizi katmandan (Linear, LeakyReLU, Dropout) oluşur. Son katmanda, kayıp fonksiyonu türüne göre Sigmoid aktivasyon kullanılabilir. Discriminator, regularization türü olarak DragGAN penalty'si uygulayabilir. Ağlar, eğitim sırasında uygun optimizer ve hiperparametreler kullanılarak optimize edilir.

**Hiperparametreler Neler?**

* embedding\_dim: Giriş katmanı boyutu.
* gen\_dim: Generator gizli katman boyutları.
* dis\_dim: Discriminator gizli katman boyutları.
* l2scale: L2 ağırlık azaltma (regularization) ölçeği.
* epochs: Eğitim dönem sayısı.
* pack: Paket boyutu.
* log\_frequency: Log frekansı.
* disabled\_dp: Diferansiyel gizlilik devre dışı bırakma.
* target\_delta: Hedef delta değeri.
* sigma: Sigma değeri (gizlilik parametresi).
* max\_per\_sample\_grad\_norm: Örnek başına maksimum gradient normu.
* verbose: Detaylı çıktı.
* loss: Kayıp fonksiyonu türü (cross\_entropy veya wasserstein).
* regularization: Regularizasyon türü (dragan).
* binary: İkili sınıflandırma.
* batch\_size: Mini-batch boyutu.
* teacher\_iters: Öğretmen discriminator eğitim iterasyonları.
* student\_iters: Öğrenci discriminator eğitim iterasyonları.
* sample\_per\_teacher: Öğretmen başına örnek sayısı.
* epsilon: Diferansiyel gizlilik parametresi epsilon.
* delta: Diferansiyel gizlilik parametresi delta.
* noise\_multiplier: Gürültü çarpanı.
* moments\_order: Moment düzeni.

**Loss Fonksiyonu Olarak Ne Kullanılmış?**

İki farklı kayıp fonksiyonu desteklenir:

* cross\_entropy: Klasik binary cross-entropy kaybı.
* wasserstein: Wasserstein kaybı (alternatif olarak w\_loss fonksiyonu ile tanımlanmış).

### Klasik GAN Yönteminden ve Varsa Temel Aldığı Değiştirilmiş GAN Versiyonundan Farkı Nedir?

**PATECTGAN** modeli, klasik GAN modeline birkaç önemli değişiklik ve ekleme yapar:

1. **Diferansiyel Gizlilik (Differential Privacy)**: PATECTGAN, diferansiyel gizliliği sağlayacak şekilde tasarlanmıştır. Bu, her öğretmen discriminator'un çıktısına gürültü ekleyerek ve gizlilik bütçesi epsilon'u takip ederek gerçekleştirilir.
2. **PATE Mekanizması**: Öğretmen ve öğrenci discriminator'lar arasındaki etkileşimde PATE (Private Aggregation of Teacher Ensembles) mekanizması kullanılır. Bu, birçok öğretmen discriminator'un çıktılarının birleştirilmesi ve bir öğrenci discriminator'un eğitilmesi ile gerçekleştirilir.
3. **Dragan Regularizasyonu**: Eğer dragan regularizasyonu seçilmişse, discriminator eğitiminde DragGAN penalty'si kullanılır. Bu penalty, gradient normlarını kontrol altında tutarak eğitim stabilitesini artırır.

**1.2. Tasarladığım Patectgan**

**Generator Ağı:**

Generator, çok katmanlı bir sinir ağıdır ve giriş olarak rasgele gürültü vektörleri alır. Her katman, LeakyReLU aktivasyon fonksiyonu ve Batch Normalizasyon ile birlikte lineer dönüşümlerden oluşur. Son katman ise Tanh aktivasyon fonksiyonunu kullanır.

**Generator Ağının Detayları :**

* **Giriş boyutu (input\_dim):** 8 (veri setindeki özellik sayısı)
* **Gizli katman boyutları (gen\_dims):** [256, 256]
* **Çıkış boyutu (output\_dim):** 8
* **Aktivasyon fonksiyonları:** LeakyReLU, Tanh
* **Düzenleme teknikleri:** Batch Normalizasyon, Dropout

**Diskriminatör Ağı :**

Diskriminatör, yine çok katmanlı bir sinir ağıdır ve giriş olarak gerçek veya generator tarafından üretilmiş verileri alır. Spectral Normalization ve LeakyReLU aktivasyon fonksiyonları ile donatılmıştır. Çıkış katmanı, sahte veriyi ayırt edebilmek için Sigmoid aktivasyon fonksiyonu kullanır.

**Diskriminatör Ağının Detayları**

* **Giriş boyutu (input\_dim):** 8 (veri setindeki özellik sayısı)
* **Gizli katman boyutları (dis\_dims):** [256, 256]
* **Paket boyutu (pack):** 1
* **Aktivasyon fonksiyonları:** LeakyReLU, Sigmoid (opsiyonel)
* **Düzenleme teknikleri:** Spectral Normalizasyon, Dropout

Oluşturduğum Model, WGAN-GP (Wasserstein GAN with Gradient Penalty) ve Özellik Eşleştirme ile optimize edilmiştir. WGAN-GP, standart GAN'lara göre daha kararlı bir eğitim sağlar ve gradyan cezası kullanarak optimizasyonu iyileştirir. Özellik Eşleştirme, generatorün ürettiği verilerin gerçek verilerin özellik dağılımlarını daha iyi taklit etmesini sağlar.

1. **Sözde Kod**

class Generator:  
 initialize(input\_dim, gen\_dims, output\_dim):  
 build\_layers(input\_dim, gen\_dims, output\_dim)  
 initialize\_weights()  
 build\_layers(input\_dim, gen\_dims, output\_dim):  
 for each dim in gen\_dims:  
 add Linear layer  
 add LeakyReLU activation  
 add Batch Normalization  
 add Dropout  
 add final Linear layer  
 add Tanh activation  
 initialize\_weights():  
 for each layer:  
 initialize weights with Xavier uniform  
 forward(input):  
 return sequentially pass input through layers  
class Discriminator:  
 initialize(input\_dim, dis\_dims, loss, pack):  
 build\_layers(input\_dim, dis\_dims, loss)  
 initialize\_weights()  
 build\_layers(input\_dim, dis\_dims, loss):  
 for each dim in dis\_dims:  
 add Spectral Normalization Linear layer  
 add LeakyReLU activation  
 add Dropout  
 add final Linear layers with Spectral Normalization  
 if loss is cross\_entropy:  
 add Sigmoid activation  
 initialize\_weights():  
 for each layer:  
 initialize weights with Xavier uniform  
  
 forward(input):  
 return sequentially pass input through layers  
 gradient\_penalty(real\_data, fake\_data, device, lambda\_):  
 calculate interpolates  
 calculate gradients  
 return gradient penalty  
class PATECTGAN:  
 initialize(params):  
 set parameters  
 initialize Generator and Discriminator  
 train(data\_loader):  
 for each epoch:  
 for each batch in data\_loader:  
 train discriminator  
 train generator  
 generate(n):  
 return generated data  
 def load\_and preprocess\_data(train\_path, test\_path):  
 load data from CSV  
 fill missing values  
 scale data  
 return scaled data  
 def evaluate\_model(model, test\_data, scaler):  
 generate data  
 calculate evaluation metrics  
 print metrics  
 main:  
 load and preprocess data  
 initialize PATECTGAN model  
 train model  
 evaluate model

1. **Algoritmanın Uygulanması**

**Ön İşleme Adımları**

**Veri Yükleme**: Eğitim ve test verileri, diabetes\_data\_train.csv ve diabetes\_data\_test.csv dosyalarından pandas kullanılarak yüklenir.

**Eksik Değerlerin Doldurulması**: Eğitim ve test veri kümelerinde bulunan eksik değerler, her sütun için ortalama değerle doldurulur.

**Veri Dönüştürme**: Veri kümeleri numpy dizilerine dönüştürülür ve veri türü float32 olarak ayarlanır.

**Özellik Ölçekleme**: Veriler StandardScaler kullanılarak ölçeklendirilir. Eğitim verileri, fit\_transform yöntemi ile ölçeklendirilirken, test verileri aynı ölçekleme ile transform yöntemi kullanılarak dönüştürülür.

**Eğitim Süreci ve Parametreleri**

**Optimizer**: AdamW optimizasyon yöntemi, hem Generator hem de Discriminator için kullanılır.

**Kayıp Fonksiyonu**: Özel olarak tanımlanan w\_loss kullanılır.

**Eğitim Döngüsü**: Model, belirlenen epoch sayısı boyunca eğitim verisi üzerinde eğitilir. Her epoch için:

**Discriminator Eğitimi**: Gerçek ve sahte veriler kullanılarak Discriminator ağı eğitilir. Kayıp fonksiyonu hesaplanır ve gradient penalty eklenir.

**Generator Eğitimi**: Sahte veriler oluşturularak Generator ağı eğitilir. Feature Matching Loss kullanılarak kayıp hesaplanır ve geri yayılım yapılır.

**Eğitim Süreci İzleme**: Eğitim süresince Discriminator ve Generator kayıpları izlenir ve her epoch sonunda ekrana yazdırılır.

### Değerlendirme Kriterleri

1. **Ortalama Kare Hata (MSE)**: Gerçek test verisi ile üretilen veriler arasındaki ortalama kare hata hesaplanır.
2. **Kolmogorov-Smirnov (KS) Testi**: Gerçek ve üretilen veriler arasındaki KS istatistiği hesaplanır.
3. **Jensen-Shannon Divergence (JSD)**: Gerçek ve üretilen veriler arasındaki Jensen-Shannon divergensi hesaplanır.
4. **Kullback-Leibler (KL) Divergence**: Gerçek ve üretilen veriler arasındaki Kullback-Leibler divergensi hesaplanır.

### Değerlendirme Süreci

1. **Veri Üretimi**: Eğitilen model kullanılarak belirtilen sayıda sahte veri üretilir.
2. **Veri Ters Dönüştürme**: Üretilen veriler, ölçekleme öncesi değerlere geri dönüştürülür.
3. **Performans Metriklerinin Hesaplanması**: Yukarıda belirtilen metrikler hesaplanarak modelin performansı değerlendirilir ve sonuçlar ekrana yazdırılır.
4. **Sonuçların Kaydedilmesi**: Değerlendirme sonuçları bir dosyaya yazdırılır.

**4. Elde Edilen Sonuç Değerleri**

**20 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 565.0940551757812 |
| Kolmogorov-Smirnov Testi | 0.3757666257799584 |
| Jensen-Shannon Divergence | 0.6517348822115212 |
| Kullback-Leibler Divergence | 4.405220891346931 |

Bu hiperparametre seti ile modelin hatası (MSE) en düşük seviyede. KS Test Statistic, veri dağılımının gerçeğe yakınlığını ölçüyor ve bu değerin nispeten düşük olması modelin iyi performans gösterdiğini belirtiyor.

**50 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 599.02197265625 |
| Kolmogorov-Smirnov Testi | 0.4078182496933497 |
| Jensen-Shannon Divergence | 0.6224813804132274 |
| Kullback-Leibler Divergence | 4.811234039637829 |

Epoch sayısı artırıldığında, MSE ve KS Test Statistic değerlerinde kötüleşme görülüyor. JSD biraz iyileşmiş olsa da, KL Divergence artışı, modelin veri dağılımındaki farklılıklarının arttığını gösteriyor.

**300 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 600.3098754882812 |
| Kolmogorov-Smirnov Testi | 0.3630206388992587 |
| Jensen-Shannon Divergence | 0.6356731538329484 |
| Kullback-Leibler Divergence | 4.984973110683873 |

Bu epoch sayısında, MSE ve JSD biraz daha kötüleşmiş durumda. Ancak, KS Test Statistic biraz iyileşmiş, bu da bazı yönlerden modelin daha iyi genel performans gösterdiğini gösteriyor.

**500 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 619.8973999023438 |
| Kolmogorov-Smirnov Testi | 0.36632712921977495 |
| Jensen-Shannon Divergence | 0.6425167521313909 |
| Kullback-Leibler Divergence | 4.026183648161274 |

Bu hiperparametre setiyle modelin MSE değeri daha da kötüleşmiş. KS Test Statistic ve JSD değerleri de biraz artmış, bu da performansın biraz daha düştüğünü gösteriyor. Ancak, KL Divergence azalmış, bu da veri dağılımının daha tutarlı olabileceğini gösteriyor.

**800 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 600.8392944335938 |
| Kolmogorov-Smirnov Testi | 0.3542744386966029 |
| Jensen-Shannon Divergence | 0.6499013234554405 |
| Kullback-Leibler Divergence | 3.912164313771429 |

Epoch sayısı 800 olduğunda, MSE ve JSD değerleri tekrar biraz kötüleşmiş. Ancak, KS Test Statistic biraz daha düşük, bu da modelin bazı yönlerden daha iyi performans gösterdiğini belirtiyor.

**1000 epoch:**

|  |  |
| --- | --- |
| İstatistiksel Ölçüt | Değer |
| Ortalama Kare Hata (MSE) | 596.74072265625 |
| Kolmogorov-Smirnov Testi | 0.3332088955255719 |
| Jensen-Shannon Divergence | 0.6464078093658971 |
| Kullback-Leibler Divergence | 3.642027130784335 |

En yüksek epoch sayısında, MSE değeri biraz iyileşmiş durumda. KS Test Statistic ve JSD biraz daha düşük, bu da modelin daha iyi performans gösterdiğini gösteriyor. KL Divergence en düşük seviyede, bu da veri dağılımındaki tutarlılığın en yüksek olduğunu belirtiyor.

**Sonuç**

Genel olarak, 20 epoch ile elde edilen sonuçlar en iyi gibi çünkü MSE en düşük ve diğer metrikler de makul seviyede. 1000 epoch gibi çok yüksek epoch sayılarında MSE ve KS Test Statistic'in bir miktar iyileşmesi, ancak JSD ve KL Divergence değerlerinin yüksek kalması, modelin performansının karışık olduğunu gösteriyor. Bu durum, aşırı öğrenme belirtisi olabilir.

Bu değerlendirmeler sonucunda, epoch sayısını belirlerken dikkatli olmak ve modelin performansını hem hata metrikleri hem de veri dağılımını karşılaştırarak analiz etmek önemlidir. 20 epoch ile elde edilen sonuçlar genel olarak en iyi performansı göstermiş gibi görünmektedir.